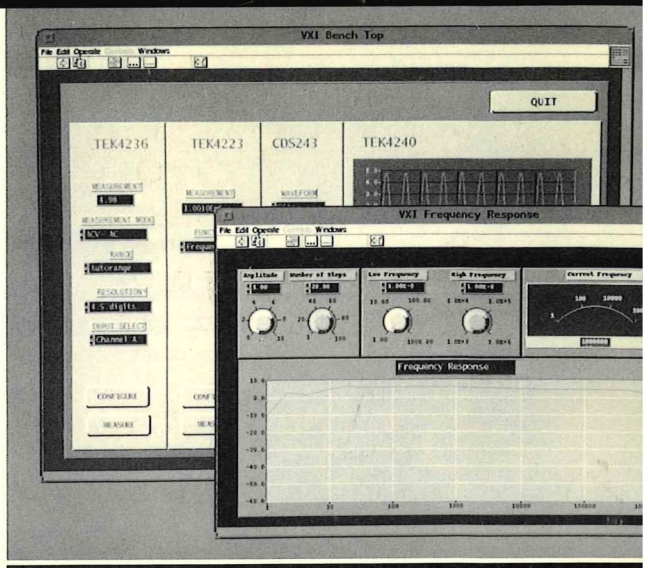
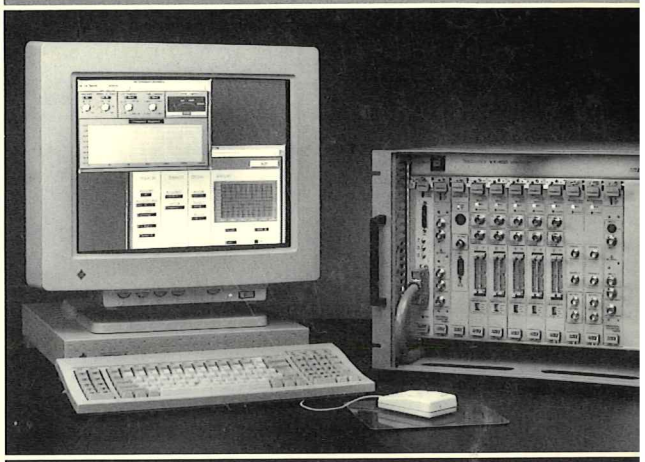
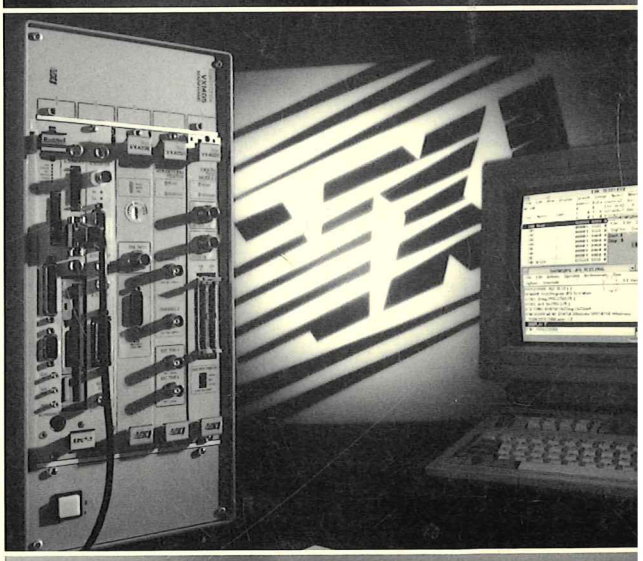
Tektronix

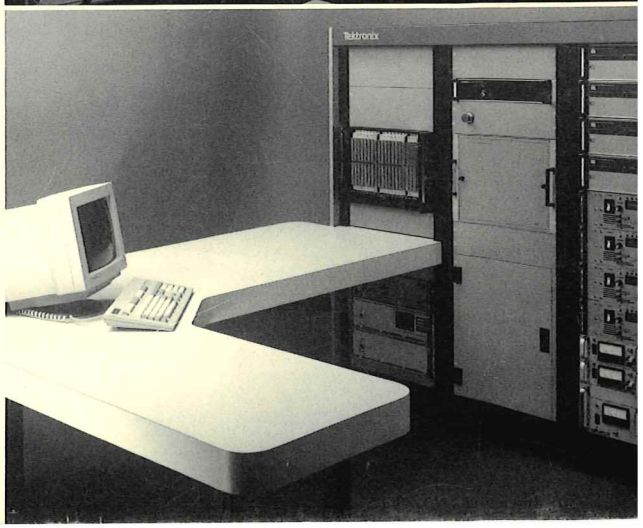
A. Schwinn

VXIbus System Specifications Revision 1.4

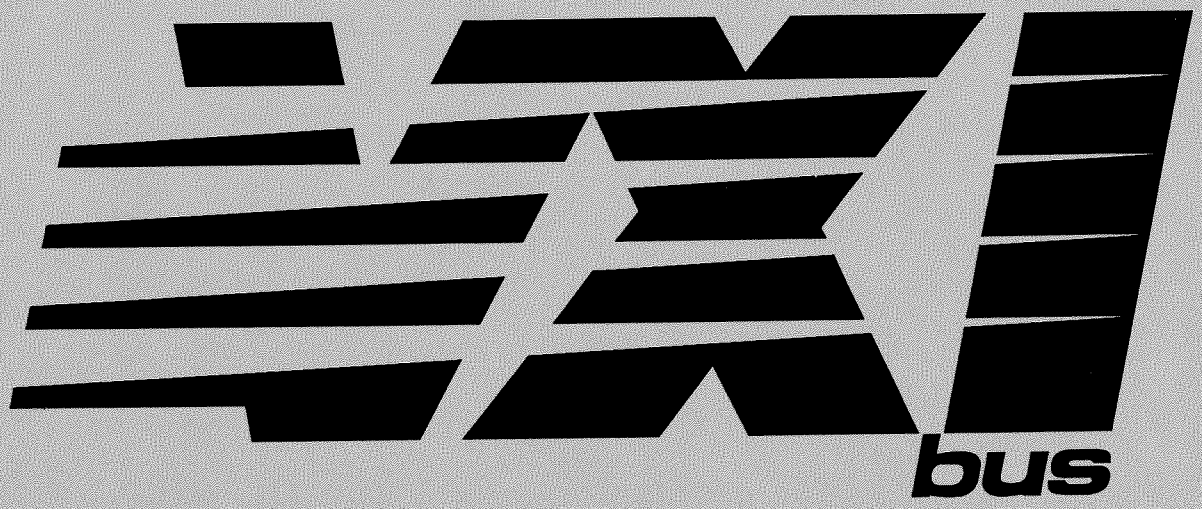








VMEbus Extensions for Instrumentation



Overview of VXIbus Specifications

VXI-0

Revision 1.0

May 22, 1992

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extensions for Instrumentation, Overview of VXIbus Specifications, VXI-0, Revision 1.0, is authored by the VXIbus Consortium, Inc. and its sponsor members:

A/S Brüel & Kjær
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments, Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce this document by any means for any purpose.

Overview of VXIbus Specifications

A. Introduction

Designers of VXIbus products are the primary audience for VXIbus specifications. The specifications are also of relevance for users of VXIbus products, but for these, detailed knowledge of the Specifications is not required. The specifications assume that the reader has an engineering level understanding of analog and digital electronics, computer bus principles, mechanical and cooling principles, and electromagnetic emissions.

This document (VXI-0) gives an overview of the specifications issued by the VXIbus Consortium. It lists current specifications, along with a brief description of their main purpose. It also points out the main relationships and dependencies between the specifications.

For each specification and revision hereof, the following information is given:

- Specification Name and Number
- Revision Number and Date
- Overview of the Specification: Contents and Purpose
- Relationships to other VXIbus specifications
- Status and Compatibility comments

Although the VXIbus Consortium makes all reasonable efforts to maintain compatibility between succeeding revisions of the VXIbus specifications, no guarantee can be issued of 100% compatibility between all specifications and different revisions of specifications. This document will note only major changes. A complete, detailed overview of changes can only be obtained by studying the individual specifications.

B. Description of VXIbus Specifications

B.1 Printing History

VXI-0: Overview of VXIbus specifications

- Rev. 1.0; May 22, 1992

VXI-1: VXIbus System Specification

- Rev. 0.0; July 9, 1987
- Rev. 1.0; August 24, 1987
- Rev. 1.1; October 7, 1987
- Rev. 1.2; June 21, 1988
- Rev. 1.3; July 14, 1989
- Rev. 1.4; April 21, 1992

VXI-2: VXIbus Extended Register Based Devices and Extended Memory Devices Specification

- Rev. 1.0; February 7, 1991

VXI-3: VXIbus Word Serial Commands for Version and Serial Number Identification Specification

- Rev. 1.0; February 7, 1991

VXI-4: VXIbus Common Mnemonics Specification

- Rev. 1.0; June 23, 1991

VXI-5: VXIbus Common ASCII System Commands Specification

- Rev. 1.0; June 23, 1991

VXI-6: VXIbus Mainframe Extender Specification

- Rev. 1.0; February 7, 1991

VXI-7: VXIbus Shared Memory Data Formats Specification

- Rev. 1.0; March 1, 1992

C. Overview of Individual Specifications

C.1 VXI-0: Overview of VXIbus Specifications, Rev. 1.0; May 22, 1992

This specification gives a complete printing history and a short description of each specification issued by the VXIbus Consortium. It is updated when significant new specifications or revisions hereof are issued.

C.2 VXI-1: VXIbus System Specification

VXI-1 is the base document upon which all current VXIbus specifications are built. The specification describes the following main subjects:

1. VXIbus Extension of the VMEbus Specification
2. Electrical Specifications of the VXIbus, Including Additions to the VMEbus
3. Module Sizes and Mechanical Dimensions
4. Mainframe Sizes and Specifications
5. EMC and Power Requirements for Modules and Mainframes
6. Suggested Test Methods for EMC and Power
7. Systems Architectures
8. Device Types: Register Based, Memory, Message Based, and Extended
9. Communication Protocols
10. Resource Manager
11. Slot 0 Functions
12. VXIbus Instruments
13. IEEE 488-VXIbus Interface
14. Command and Event Formats
15. Dynamic Configuration of VXIbus Devices

C.2.1 VXI-1, Rev. 1.0; August 24, 1987

This is the first Revision of the VXIbus specification by which working prototypes could be built. Early development work, based on this draft, pointed out a number of necessary revisions, which were incorporated in Rev. 1.1, issued two months later.

C.2.2 VXI-1, Rev. 1.1; October 7, 1987

This is the first Revision of the VXIbus specification by which early products could be built. These products showed the need for additional fine tuning of the specification, which resulted in Revision 1.2. To the best knowledge of the VXIbus Consortium, there are no Revision 1.1 products on the market.

This Revision of the specification is obsolete and the VXIbus Consortium does not recommend building products to it.

C.2.3 VXI-1, Rev. 1.2; June 21, 1988

This Revision represented a stable document upon which a number of VXIbus products have been built. Existing Revision 1.2 devices can normally successfully be supported, but it is not recommended to design new products based on this revision of the specification.

The Shared Memory Protocol described in the specification has subsequently been removed. This Revision of the specification is obsolete and the VXIbus Consortium does not recommend building products to it.

C.2.4 VXI-1, Rev. 1.3; July 14, 1989

Revision 1.3 is a stable platform on which numerous products have been built and successfully used together. Existing Revision 1.3 devices can normally be supported successfully.

This Revision of the specification is, however, obsolete and the VXIbus Consortium does not recommend building products to it.

The following is a summary of the major changes from Rev. 1.2:

1. The Device Initialization, Self-Test and Configuration sections were partially rewritten for greater clarity and robustness. Several new states as well as a new definition of the Ready and Passed bits were added. All devices end up in a deactivated, inactive state until they are explicitly configured otherwise.
2. The configuration of the Message Based devices was changed for greater clarity and robustness. Several new sub-states were added in the passed state.
3. The Word Serial Protocol was changed for greater robustness. A number of word serial protocol commands that were optional in Revision 1.2 are mandatory in Revision 1.3. Several word serial protocol commands were changed to return status information for easier error handling. A layered error handling was also defined. Possible violations of the Word Serial Protocol were defined with associated error messages.
4. The Resource Manager function was clarified and more algorithms added. Appendix III of Revision 1.3 points out the considerations a Resource Manager should take to give maximum compatibility with Revision 1.2 devices.
5. The Shared Memory Protocol of Revision 1.2 was removed.
6. Mechanical Dimensions to account for board warpage and tolerance stack-up were modified.

7. Provision for shielding around backplane connectors was added.
8. Terminology for AC Induced Current and AC Load Current was changed to reflect that this is a specification for the mainframe.
9. Minor rewrites to clarify EMC Test Requirements.

C.2.5 VXI-1, Rev. 1.4; April 21, 1992

Revision 1.4 is the most recently approved revision of VXI-1. The VXIbus Consortium recommends that all new products should be built to this specification.

This revision clarifies a number of items in the specification and adds additional requirements for the following subjects: Backplane component height, trigger protocol timing, the ECL trigger interface, connector shielding, conducted EMC, fast handshake protocol, error handling and dynamic configuration. Every effort has been made to ensure that products built to Revision 1.3 will be compatible with Revision 1.4. However, some minor incompatibilities may occur with respect to the implementation of the Fast Handshake Protocol. It was not possible to make the Fast Handshake Protocol 100% compatible, and at the same time ensure a robust implementation.

The following list gives an overview of the changes from Rev. 1.3:

1. Power Monitor (Section B.5.1): A section is added explaining the background and purpose of the monitor.
2. TTLTRG* Semi-Synchronous Trigger Protocol has been removed from the Specification.
3. ECLTRG Interface: The rules to use Schottky diodes and associated capacitors are removed and replaced by a series 150 ohm resistor. (Section 6.2.4 and Figure B.10).
4. ECL SEMI-SYNC Trigger Protocol has been removed from the Specification.
5. Timing requirements for setup and hold times for SYNC100: The Max setup time for SYNC100 is reduced from 8.0 ns to 7.5 ns, and the Max hold time for SYNC100 is reduced from 8.0 to 6.0 ns. (Figure B.15).
6. Component height on backplane: The height of conducting components is reduced from 12.2 mm to 10.0 mm. (Section B.7.3.1 Rule B.7.44)
7. Backplane Connector Shield: Additional rules and observations, and mechanical tolerances are added to Figure B.36. Certain mechanical dimensions are also modified.
8. Module Extraction Surface: Mechanical dimensions are added to Figure B.39.
9. Module Injection Surface: Dimensions referenced center line are added to Figure B.40.
10. DC Voltage Specifications: The DC Load Ripple/Noise and Induced Ripple/Noise are given new abbreviations and the use of these two parameters in specifying the total peak to peak noise in Rule B.8.6 of Section B.8.4 is more clearly defined. This is also reflected in the associated figures, Figure B.45 and B.46. In addition, the tolerance limits in Figure B.45 (Mainframe Load Current) and B.46 (Mainframe Induced and Load Ripple/Noise Limits) are changed significantly, and are generally more liberal than in Rev. 1.3.
11. Conducted Susceptibility: The new abbreviations for DC Load Ripple/Noise and induced Ripple Noise are also used in Rule B.8.12 and in the associated figures, Figures B.47 and B.48. In addition, the limits for Module Conducted Emissions (Figure B.47) and Module Susceptibility Level (Figure B.48) are changed significantly, and are generally more liberal than in Rev. 1.3.
12. Power and conducted EMC requirements: The lower frequency limit has been raised from 2 Hz to 20 Hz

(Sections B.8.4 through B.8.6 and Figures B.45 through B.48).

13. Conducted Emissions Test of Modules: Minor changes in test methods in Section B.8.7.2.
14. Address Modifier Codes: Clarifications of the functions in Section C.2.1.1.5.
15. Message Based Device Communication Registers (Section C.2.4.3.1): Addition of a prohibition for a device clearing its Read Ready or Write Ready bits at specified times. (Rules C.2.50 and C.2.51). Addition of a new rule (C.2.52) that a message based device retains the state of bits 13 - 8 in its Response register whenever the device modifies any other Response register bit.
16. Tolerance for Message Based Devices with long start-up or boot times: An Observation (C.2.72) has been added to Section C.2.4.4. (Message Based Device Operation) that states that there is no upper limit on the start-up time for certain devices, and therefore the Resource Managers should handle these tolerantly.
17. Servant Sourcing Data: A prohibition against a Servant autonomously sourcing data to its Read registers is added (Rule C.3.2 in Section C.3.3.1 Word Serial Protocols). Error handling related to this is also added (Observation C.3.1).
18. Fast Handshake Transfers and Byte Transfer Protocols (Section C.3.3.2 and C.3.3.3): Significant parts of these sections have been re-written to give better defined, more robust operation and error handling (Section C.3.3.4).
19. Detection of Failed Devices: New Permissions (C.4.3 and C.4.4) and rules Rule C.4.7 are added that permit the Resource Manager to detect and reset devices which have failed after the initial 5 second self-test period. (Section C.4.1.2 System Self Test Management).
20. Procedure for Address Map Configuration: A detailed procedure is added that defines precisely how the A24 and A32 address Map shall be configured. In addition, recommended placements for A24 and A32 registers are added (Section C.4.1.3 Address Map Configuration).
21. Prohibition against changing state of Servants: A new rule (C.4.14) states that a Resource manager is only permitted to change the configuration or state of a device's servant under a few, specified conditions (Section C.4.1.6 Initiating Normal Operation).
22. Clarification of handling of Read Protocol Error (Section E.1 Word Serial Commands).
23. Clarification of Dynamic Configuration procedures (Sections F.2 and F.3).

C.3 VXI-2: VXIbus Extended Register Based Devices and Extended Memory Devices Specification, Rev. 1.0; February 7, 1991

This specification defines a set of extra capabilities for a class of Extended Devices. For Extended Register Based Devices, the new functions are Version Number (both for firmware and hardware), and a 32-bit serial number. Also added is Interrupt Configuration capability and Interrupt Status readout.

Similar capabilities are also defined for Extended Memory Based Devices.

The considerations that the Resource Manager should take when handling these types of devices are also described.

C.4 VXI-3: VXIbus Word Serial Commands for Version and Serial Number Identification Specification, Rev. 1.0; February 7, 1991

This brief specification defines three optional commands for Message Based Devices: Read Serial Number High, Read Serial Number Low, and Read Version Number. These enable Message Based Devices to implement functionality similar to Extended Devices as defined in VXI-2.

C.5 VXI-4: VXIbus Common Mnemonics Specification, Rev. 1.0; June 23, 1991

This specification lists 151 abbreviations of the most commonly used VXIbus related terms. The abbreviations have been made compatible with IEEE 488.2 and have also been reviewed to ensure compatibility with the specifications of the SCPI Consortium.

C.6 VXI-5: VXIbus Common ASCII System Commands Specification, Rev. 1.0; June 23, 1991

This specification defines a set of ASCII commands which are issued to a VXIbus system from an external host. The commands are used to access system information, including configuration and common capability status. The terminology used is patterned after IEEE 488.2 and has also been reviewed to ensure compatibility with the specifications established by the SCPI Consortium.

The commands included are specification of logical address and configuration information from the resource manager regarding all VXIbus devices, including the commander/servant hierarchy. In addition, commands are included to read VXIbus-defined registers, to send specified word serial commands, and to read responses to these commands.

C.7 VXI-6: VXIbus Mainframe Extender Specification, Rev. 1.0; February 7, 1991

This specification defines the logical requirements for extending the VXIbus between multiple mainframes. Included in the scope of the specification are Address and Data Transfer mechanisms, Interrupts and Interrupt Handling, Inter-device Synchronization via the TTL and ECL triggers, use of the Utility Bus (ACFAIL*, SYSFAIL*, and SYSRESET*), System Configuration of Multiple Mainframes, and Multi-mainframe Resource Management.

VXI-6 does not define extensions for the following VMEbus signals: SERCLK, SERDAT, and SYSCLK. Likewise, extensions for the following VXIbus signals are not defined: SUMBUS, STARX/Y, CLK100, and SYNC 100.

The specification does not define the physical media for connecting the mainframes. Conformance to the specification requires implementation of the VXIbus address and data path extensions, while support of interrupts, triggers, and utility bus is optional.

C.8 VXI-7: VXIbus Shared Memory Data Formats Specification, Rev. 1.0; March 1, 1992

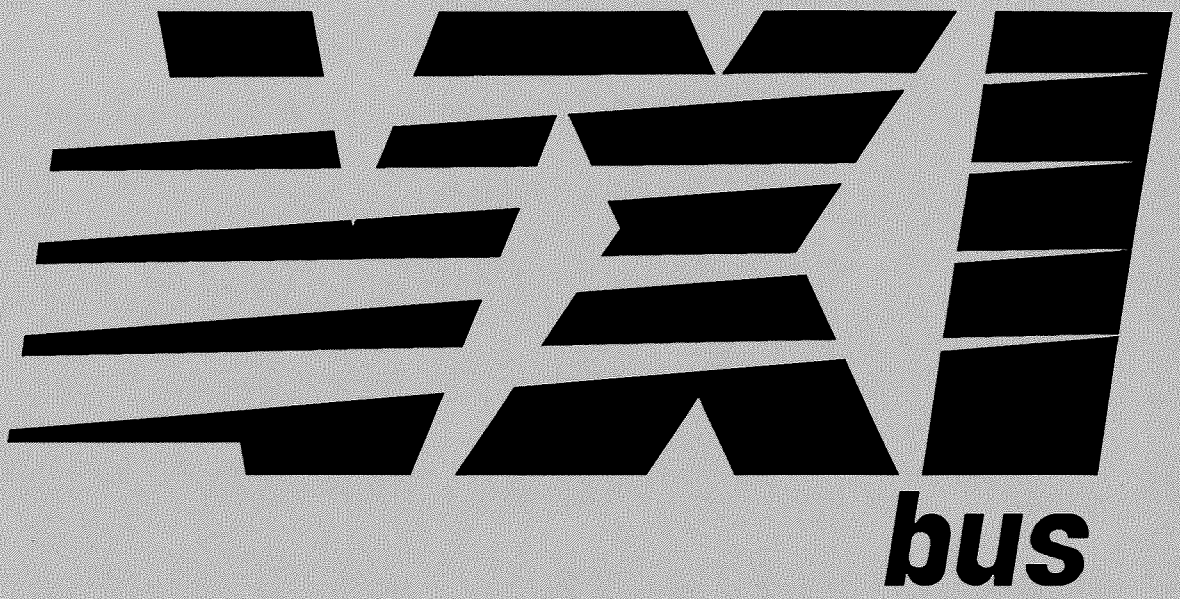
This specification defines a set of VXIbus data formats for interchange of data via the global address space. Included are data access conventions, definition of low level data element formats, and common terminology for creating more complex data structures.

Outside the scope of the specification is the definition of actual data transfer protocols.

The specification applies commonly accepted industry standards, including IEEE 754-1985, and ANSI standards for "ASCII". Data ordering and byte ordering follows the conventions common in the Motorola 68000 family of microprocessors.

The numeric formats include 8, 16, and 32-bit signed and unsigned integers, and 32 and 64 bit reals (floating point). Also included are 7 and 8-bit ASCII character formats. Arbitrary elements and data structures (arrays and strings) are also defined.

VMEbus Extensions for Instrumentation



System Specification

VXI-1

REVISION: 1.4

April 21, 1992

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extensions for Instrumentation System Specification VXI-1, Revision 1.4 is authored by the VXIbus Consortium, Inc. and its sponsor members:

Brüel & Kjær Test Systems A/S
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments, Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce and distribute this document by any means for any purpose.

VXI-1 Revision History

Revision 1.0, August 24,1987
Original issue.

Revision 1.1, October 7, 1987
Various corrections and modifications.

Revision 1.2, June 21, 1988
Additional requirements for radiated EMC, system power management, word-serial protocols, and configuration of interrupts and signals. Defined extended devices and message-based slot 0 devices. Added Dynamic Configuration and Shared Memory Communication Protocol.

Revision 1.3, July 14, 1989
Additional requirements for device initialization, word-serial protocol, and system configuration. Deleted shared memory communication protocol.

Revision 1.4, April 21, 1992
Incorporates various clarifications as well as additional requirements for backplane component height, trigger protocol timing, the ECL trigger interface, connector shielding, conducted EMC, fast handshake protocol, error handling and dynamic configuration. The Semi-Sync TTLTRG* and ECLTRG* trigger protocols are deleted. The following sections and figures are affected:

B.5.1, "Power Monitor"
B.6.2.4, "ECLTRG0-1 (ECL Trigger Lines)"
Figure B.9, "Typical ECLTRG Interface"
Figure B.13, "CLK100, SYNC100, and ECLTRG START/STOP Timing"
B.7.3.1, "Backplanes"
Figure B.34, "Backplane Connector Shield"
Figure B.37, "Module Extraction Surface"
Figure B.38, "Module Injection Surface"
B.8.4, "DC Voltage Specifications"
Figure B.43, "Mainframe Load Current"
Figure B.44, "Mainframe Induced and Load Ripple/Noise Voltage Limits"
B.8.6.2, "Conducted Susceptibility"
Figure B.45, "Module Conducted Emissions"
Figure B.46, "Module Susceptibility Level"
B.8.7.2, "Conducted Emissions Test of Modules"
C.2.1.1.5, "Address Modifiers"
C.2.4.3.1, "Message Based Device Communication Registers"
C.2.4.4, "Message Based Device Operation"
C.3.3.1, "Word Serial Protocols"
C.3.3.2, "Fast Handshake Transfers"
C.3.3.3, "Byte Transfer Protocol"
C.3.3.4, "Error Handling"
C.4.1, "Resource Manager"
E.1, "Word Serial Commands"
F.2, "DC Device Requirements"
F.3, "DC System Requirements"

CONTENTS

A. INTRODUCTION to VXIbus SPECIFICATIONS: 1

A.1 MANUFACTURER ID NUMBERS 2

A.2 VXIbus OVERVIEW 3

A.2.1 Introduction 3

A.2.2 VMEbus Background 3

A.2.3 The VXIbus Extensions 4
A.2.3.1 VXIbus MODULES 4
A.2.3.2 VXIbus SUBSYSTEMS 4
A.2.3.3 VXIbus SYSTEM ARCHITECTURE 5
A.2.3.4 VXIbus SPECIFICATION STRUCTURE 5

A.3 DOCUMENT STRUCTURE 6

A.4 SPECIFICATION OBJECTIVES 6

A.5 DEFINITION of TERMS 7

B. VXIbus IMPLEMENTATION of VMEbus SPECS: 8

B.1 INTRODUCTION 8

B.2 DATA TRANSFER BUS 8

B.2.1 Address Modifiers 8

B.2.2 DTACK* Operation 8

B.2.3 Bus Timer Operation 8

B.3 DATA TRANSFER BUS ARBITRATION 9

B.4 PRIORITY INTERRUPT 9

B.5 UTILITIES 9

B.5.1 Power Monitor 9

B.5.2 Power Pins 9

B.6 ELECTRICAL SPECIFICATIONS 10

B.6.1 The P1 Connector 10

B.6.2 The VXIbus Subsystem P2 Connector 10
B.6.2.1 CLK10 10
B.6.2.2 MODID LINES 14
B.6.2.3 TTLTRG0-7* (TTL TRIGGER LINES) 15
B.6.2.4 ECLTRG0-1 (ECL TRIGGER LINES) 22
B.6.2.5 SUMBUS 27
B.6.2.6 LOCAL BUS 29
B.6.2.7 RSV2-3 (RESERVED) 31

B.6.3 The VXIbus Subsystem P3 Connector 31
B.6.3.1 CLK100 31
B.6.3.2 SYNC100 34
B.6.3.3 STARX and STARY 35

B.6.3.4 ECLTRG2-5 36
B.6.3.5 LBUS12-35 36
B.6.3.6 RSV4-7 36

B.6.4 Backplane 36
B.6.4.1 ECL SIGNAL LEVELS 37

B.7 MECHANICAL SPECIFICATIONS 38

B.7.1 Introduction 38

B.7.2 Module Specifications 38
B.7.2.1 VXIbus BOARDS and MODULES 38
B.7.2.2 FRONT PANEL 40
B.7.2.3 MODULE SHIELDING 43
B.7.2.4 MODULE COOLING 44
B.7.2.5 MODULE POWER 45
B.7.2.6 MODULE KEYING 45
B.7.2.7 MODULE ENVIRONMENTAL 45

B.7.3 Mainframe Specifications 45
B.7.3.1 BACKPLANES 47
B.7.3.2 GROUNDING 47
B.7.3.3 INJECTION/EJECTION 48
B.7.3.4 MAINFRAME SHIELDING 48
B.7.3.5 MAINFRAME COOLING 48
B.7.3.6 MAINFRAME POWER 49
B.7.3.7 KEYING 49
B.7.3.8 MAINFRAME ENVIRONMENTAL 49

B.8 EMC and SYSTEM POWER 79

B.8.1 Introduction 79

B.8.2 Power Distribution 79

B.8.3 Power Pins 79

B.8.4 DC Voltage Specifications 79

B.8.5 Power Management 82

B.8.6 Electromagnetic Compatibility (EMC) of Modules 83
B.8.6.1 CONDUCTED EMISSIONS 83
B.8.6.2 CONDUCTED SUSCEPTIBILITY 83
B.8.6.3 RADIATED EMISSIONS 85
B.8.6.4 RADIATED SUSCEPTIBILITY 87

B.8.7 Suggested Test Methods 94
B.8.7.1 DC LOAD RIPPLE/NOISE TEST OF MAINFRAMES 94
B.8.7.2 INDUCED RIPPLE/NOISE TEST OF MAINFRAMES 94
B.8.7.3 CONDUCTED EMISSIONS TEST OF MODULES 94
B.8.7.4 CONDUCTED SUSCEPTIBILITY TEST OF MODULES 95
B.8.7.5 CLOSE-FIELD MAGNETIC EMISSIONS TEST 95

B.8.7.6 CLOSE-FIELD MAGNETIC SUSCEPTIBILITY TEST 95
B.8.7.7 EXAMPLES OF CLOSE FIELD MAGNETIC PROBES 95

C. SYSTEM ARCHITECTURE: 96

C.1 VXIbus SYSTEM ARCHITECTURE OVERVIEW 96

C.2 DEVICE OPERATION 99

C.2.1 Device Overview 99
C.2.1.1 DEVICE SLAVE CAPABILITIES 99
C.2.1.2 DEVICE INITIALIZATION and DIAGNOSTICS 107
C.2.1.3 PRIORITY INTERRUPTS 112
C.2.1.4 VMEbus MASTER CAPABILITIES 114
C.2.1.5 TERMINATING OPERATION 115

C.2.2 Register Based Devices 117
C.2.2.1 DATA TRANSFER CAPABILITIES 117
C.2.2.2 PRIORITY INTERRUPTS 117
C.2.2.3 REGISTER BASED DEVICE REGISTERS 118
C.2.2.4 TERMINATING OPERATION 118

C.2.3 Memory Devices 120
C.2.3.1 MEMORY DEVICE REGISTERS 120

C.2.4 Message Based Devices 122
C.2.4.1 DATA TRANSFER CAPABILITIES 122
C.2.4.2 PRIORITY INTERRUPTS 123
C.2.4.3 MESSAGE BASED DEVICE REGISTERS 125
C.2.4.4 MESSAGE BASED DEVICE OPERATION 130
C.2.4.5 MESSAGE BASED DEVICE CONFIGURATION 135
C.2.4.6 INITIATING OPERATION 137
C.2.4.7 TERMINATING OPERATION 139
C.2.4.8 CLEARING A MESSAGE BASED DEVICE 143

C.2.5 Extended Devices 144

C.3 DEVICE COMMUNICATION PROTOCOLS: 146

C.3.1 Communication Elements 146
C.3.1.1 REGISTER BASED SERVANTS 146
C.3.1.2 MESSAGE BASED SERVANTS 146
C.3.1.3 MESSAGE BASED COMMANDERS 146

C.3.2 Register Based Servant Control 146

C.3.3 Message Based Servant Control 146
C.3.3.1 WORD SERIAL PROTOCOLS 147
C.3.3.2 FAST HANDSHAKE TRANSFERS 149
C.3.3.3 BYTE TRANSFER PROTOCOL 151
C.3.3.4 ERROR HANDLING 154
C.3.3.5 DEVICE FAILURES 155

C.4 SYSTEM RESOURCES 157

C.4.1 Resource Manager 157
C.4.1.1 DEVICE IDENTIFICATION 158

C.4.1.2 SYSTEM SELF TEST MANAGEMENT 158
C.4.1.3 ADDRESS MAP CONFIGURATION 159
C.4.1.4 COMMANDER/SERVANT HIERARCHIES 159
C.4.1.5 ALLOCATING IRQ LINES 160
C.4.1.6 INITIATING NORMAL OPERATION 161

C.4.2 Runtime Resource Management 162

C.4.3 VXIbus Subsystem Slot 0 163
C.4.3.1 REGISTER BASED SLOT 0 DEVICES 164
C.4.3.2 MESSAGE BASED SLOT 0 DEVICES 164
C.4.3.3 OTHER SLOT 0 DEVICES 165

D. VXIbus DEVICE IMPLEMENTATIONS 166

D.1 VXIbus INSTRUMENTS 166

D.1.1 VXIbus Instrument Protocols 166
D.1.1.1 DATA TRANSFER from COMMANDERS to INSTRUMENTS 166
D.1.1.2 DATA TRANSFER from INSTRUMENTS to COMMANDERS 167
D.1.1.3 CLEARING a VXIbus INSTRUMENT 167
D.1.1.4 TRIGGERING an INSTRUMENT 168
D.1.1.5 LOCAL LOCKOUT 168
D.1.1.6 SRQ OPERATION 169
D.1.1.7 SPOLL OPERATION 169
D.1.1.8 ERROR REPORTING 169
D.1.1.9 INITIALIZATION 170

D.1.2 VXIbus IEEE-488.2 Instrument Protocols 170
D.1.2.1 CLEARING a VXIbus IEEE-488.2 INSTRUMENT 170
D.1.2.2 TRIGGERING an IEEE-488.2 INSTRUMENT 170
D.1.2.3 LOCAL LOCKOUT 170
D.1.2.4 SRQ OPERATION 171
D.1.2.5 SPOLL OPERATION 171

D.2 488-VXIbus INTERFACE 172

D.2.1 IEEE 488 Address Mapping 172

D.2.2 488-VXIbus Interface Device to IEEE 488 Bus Functions. 173

D.2.3 VXIbus Instrument Protocol 173
D.2.3.1 DATA TRANSFER from INTERFACE DEVICE to VXIbus INSTRUMENT 174
D.2.3.2 DATA TRANSFER from VXIbus INSTRUMENT to VXIbus INTERFACE DEVICE 176
D.2.3.3 DEVICE CLEAR OPERATION 176
D.2.3.4 TRIGGER OPERATION 176
D.2.3.5 IEEE 488 REMOTE LOCAL 177
D.2.3.6 SRQ OPERATION 177
D.2.3.7 SPOLL Operation 178

E. COMMAND and EVENT FORMATS 179

E.1 WORD SERIAL COMMANDS 179

E.2 LONGWORD SERIAL COMMANDS 198

E.3 EXTENDED LONGWORD SERIAL COMMANDS 199
E.4 PROTOCOL EVENTS 200
F. DYNAMIC CONFIGURATION 201
F.1 DEFINITIONS 201
F.2 DC DEVICE REQUIREMENTS 201
F.2.1 Logical Address Register 201
F.2.2 DC Device Logical Address Assignment 202
F.2.3 Offset Register 202
F.2.4 MODID Support 203
F.3 DC SYSTEM REQUIREMENTS 204
F.3.1 System Configuration Algorithm 204
F.3.1.1 SC DEVICE IDENTIFICATION 205
F.3.1.2 DC DEVICE LOGICAL ADDRESS ASSIGNMENT 205
F.3.1.3 HIERARCHY CONSTRUCTION 206
G. SHARED MEMORY PROTOCOLS 207
APPENDIX I. VXIbus REGISTER OVERVIEWS 208
APPENDIX II. SUGGESTED BACKPLANE DESIGN. 218
II.1 BACKPLANE STRUCTURE 218
II.2 BACKPLANE LAYOUT 218
APPENDIX III. Support of Earlier VXIbus Revisions 222
III.1 REVISION 1.2 DEVICES 222
III.2 REVISION 1.3 DEVICES 223
APPENDIX IV. GLOSSARY OF TERMS 224
REFERENCES 227

LIST OF FIGURES

Figure B.1. CLK10, MODID, and LBUS backplane signal routing. 13
Figure B.2. Module ID lines. 15
Figure B.3. TTLTRG* Synchronous (SYNC) Trigger Protocol. 17
Figure B.4. TTLTRG* Asynchronous (ASYNC) Trigger Protocol. 18
Figure B.5. TTLTRG* Data Transmission on falling clock edge. 19
Figure B.6. TTLTRG* Data Transmission on rising clock edge. 19
Figure B.7. TTLTRG* START/STOP timing. 20
Figure B.8. External Trigger Buffering 21
Figure B.9. Typical ECLTRG Interface. 23
Figure B.10. ECLTRG Synchronous (SYNC) Trigger Protocol. 24
Figure B.11. ECLTRG Asynchronous (ASYNC) Trigger Protocol. 25
Figure B.12. ECLTRG Data Transmission. 26
Figure B.13. CLK100, SYNC100, and ECLTRG START/STOP Timing. 28
Figure B.14. Typical D-size mainframe. 52
Figure B.15. Size C board 53
Figure B.16. Size D board 54
Figure B.17. Module envelope, top view 55
Figure B.18. Module envelope, front view 56
Figure B.19. C-size board assembly connector positions. 57
Figure B.20. D-size board assembly connector positions. 58
Figure B.21. Module edge guide feature. 59
Figure B.22. C-size Front panel details. 60
Figure B.23. D-size Front panel details. 61
Figure B.24. C-size filler panel. 62
Figure B.25. D-size filler panel. 63
Figure B.26. Typical C-size front panel mounting and dimensions. 64
Figure B.27. Typical D-size front panel mounting and dimensions. 65
Figure B.28. Local bus lockout key details. 66
Figure B.29. C-size mainframe drawing 67
Figure B.30. D-size mainframe drawing 68
Figure B.31. C-size backplane drawing. 69

Figure B.32. D-size backplane drawing. 70
Figure B.33. Detailed backplane dimensions. 71
Figure B.34. Backplane connector shield. 72
Figure B.35. C-size board guide detail 73
Figure B.36. D-size board guide detail 74
Figure B.37. Module extraction surface. 74
Figure B.38. Module injection surface. 74
Figure B.39. C and D size module inlet and exhaust areas. 76
Figure B.40. Simplified test for module cooling specifications. 77
Figure B.41. Simplified test for mainframe cooling specifications. 78
Figure B.42. Example of ΔP versus airflow for system integration. 79
Figure B.43. Mainframe Load Current 81
Figure B.44. Mainframe Induced and Load Ripple/Noise Voltage Limits 81
Figure B.45. Module Conducted Emissions 84
Figure B.46. Module Susceptibility Level 84
Figure B.47. A & B size Maximum close-field emissions (dB above 1 picotesla). 85
Figure B.48. C & D size Maximum close-field emissions (dB above 1 picotesla). 86
Figure B.49. A & B size Minimum close-field susceptibility (dB above 1 picotesla). 88
Figure B.50. C & D size Minimum close-field susceptibility (dB above 1 picotesla). 88
Figure B.51. A & B size EMC Test Areas. 91
Figure B.52. C & D size EMC Test Areas. 92
Figure B.53. Adapted A & B size & C & D size EMC Test Areas. 93
Figure B.54. Adapted A & B size & C & D size EMC Test Areas. 94
Figure C.1. Typical System Configurations. 96
Figure C.2. VXIbus Communication Layers 97
Figure C.3. Device Classifications. 100
Figure C.4. Self Test State Diagram 109
Figure C.5. Configure State Diagram 131
Figure D.1. VXIbus Instrument Protocol 174
Figure D.2. Message Exchange Control Interface Functional Blocks 175
Figure I.1. Register Map for Register Based Devices. 210
Figure I.2. Register Map for Memory Devices. 211
Figure I.3. Register Map for Message Based Devices. 212
Figure I.4. Register Map for Extended Devices. 213
Figure I.5. Register Overview for Basic Configuration Registers. 214

Figure I.6. Register Overview for MODID Register in Register Based Slot 0 Devices. 215
Figure I.7. Register Overview for Attribute Register in Memory Devices. 216
Figure I.8. Register Overview for Subclass Register in Extended Devices. 217
Figure I.9. Register Overview for Communication Registers. 218
Figure II.1. Typical Logic Circuits. 219
Figure II.2. Loop Area With Cut in Ground Plane. 220
Figure II.3. Backplane Cutouts for Connectors. 221

LIST OF TABLES

TABLE B.1. P2 Pin Definitions: Slots 1-12. 11
TABLE B.2. P2 Pin Definitions: Slot 0. 12
TABLE B.3. Logical Classes of LBUS Signals. 29
TABLE B.4. P3 Pin Definitions. 32
TABLE B.5. P3 Slot 0 Pin Definitions. 33
TABLE B.6. Backplane ECL signal levels. 37
TABLE B.7. Power Supply Voltage Specifications 80
TABLE C.1. The Default Configuration. 136
TABLE C.2. Message Based Device Termination and Clear Actions 139
TABLE E.1. General Command Requirements 179
TABLE E.2. Asynchronous Command Requirements 180

VMEbus Extensions for Instrumentation

System Specification

A. INTRODUCTION to VXIbus SPECIFICATIONS:

This document discusses the VXIbus Family of VMEbus compatible modular instrumentation. It is intended to be used by designers interested in generating compatible components for the system.

The architectural concepts of the VMEbus date back to Motorola's development of the 68000 microprocessor in the late 1970's. In late 1979 Motorola published a brief description of a 68000 oriented bus known as VERSAbus. Several revisions followed, the last being in July 1981.

At the same time, a new printed circuit board standard (IEC 297-3) was being developed, known as the "Eurocard" standard. In October 1981, Motorola, Mostek, and Signetics announced their agreement to support a line of cards based on the VERSAbus with Eurocard module dimensions, which was renamed VMEbus. There have been three releases of the VMEbus specifications, with the most recent, Revision C.1, published in April 1985. The VMEbus was recently standardized as IEEE 1014.

The marketplace has demonstrated the open system nature of VMEbus. There are thousands of VMEbus cards available from a multitude of vendors. These cards are primarily computer oriented cards, though there are some data acquisition cards available which are primarily aimed at industrial processing and control.

There have been numerous requests for instrumentation modules for VMEbus, particularly from the United States Department of Defense. Reduction of the size of ATE equipment is a major factor driving the Defense Department toward VMEbus, though the high bandwidth of the VMEbus backplane is also considered a benefit, particularly for digital test and digital signal processing applications. The United States Air Force, Navy, and Army have programs addressing this issue, known as MATE, CASS, and IFTE respectively.

The most serious impediment to VMEbus based instrumentation is the lack of standards beyond that of VMEbus. In the spring of 1987, technical representatives from Colorado Data Systems, Hewlett Packard, Racal Dana, Tektronix, and Wavetek formed an ad hoc committee to engineer the additional standards necessary for an open architecture instrumentation bus based on VMEbus, the Eurocard standard, and other instrumentation standards such as IEEE 488.2. In July of 1987 they announced their agreement to support a common architecture of VMEbus Modular Instrumentation, named the VXIbus.

A.1 MANUFACTURER ID NUMBERS

The VXIbus specification provides a mechanism for identifying the manufacturer of every VXIbus device. This consists of a 12 bit manufacturer ID number (0 → 4095) which may be read over the VMEbus (See Section C.2.1.1.2, "Configuration Registers"). The list of ID numbers is maintained by the VXIbus Consortium, and is available to the public upon request. Each VXIbus device manufacturer has exactly one Manufacturer ID number. Numbers are assigned to manufacturers in decreasing order beginning with number 4095. To obtain an ID number, submit the following application:

Application for Obtaining a VXIbus Manufacturer's Identification Number

Company Name: _____ Date: _____

Company Contact: _____

Title or Position: _____

Address : _____

_____

Phone Number : _____

Mail to the VXIbus Consortium:

VXIbus Consortium, Inc.
c/o VITA
10229 N. Scottsdale Road, Suite B
Scottsdale, AZ 85253-1437

Telephone: (602) 951-8866

Fax: (602) 951-0720

A.2 VXIbus OVERVIEW

A.2.1 Introduction

The goal of the VXIbus is to define a technically sound modular instrument standard based on the VMEbus that is open to all manufacturers and is compatible with present industry standards.

VXIbus is an acronym for VMEbus extensions for Instrumentation. The VXIbus specification details the technical requirements of VXIbus compatible components, such as mainframes, backplanes, power supplies, and modules. Before studying the VXIbus architecture, one should become familiar with the VMEbus and its specification.

A.2.2 VMEbus Background

The VMEbus is an open system architecture primarily focused at computer systems, though there presently is a limited offering of instrumentation. VMEbus modules are approximately six inches deep and come in two heights, about four inches and nine inches. The VXIbus specification refers to these as the A and B sizes respectively. The precise dimensions are specified by the Eurocard standard, which describes a family of printed circuit boards and their associated DIN connector locations. VMEbus modules are designed for 0.8 inch slot to slot spacing. The A size board has a single 96 pin connector known as P1, while the B size may include a P1 and P2 connector. Each of these DIN connectors consists of three rows of 32 pins apiece on 0.1 inch centers. Typically, these boards are positioned vertically in a frame with the P1 connector closest to the top. Neither the VMEbus nor the VXIbus mandates a physical orientation, since orientation is only an implementation issue not needed for compatibility. Many VMEbus systems are designed to accept boards horizontally.

The VMEbus specification allows a maximum of 21 modules. However, if installed vertically in a mainframe intended for mounting in a standard 19 inch rack, 20 is the practical maximum. VMEbus makes no particular provision for an extension chassis or frame to frame communication. Multiple frame systems can be created by electrically buffering the VMEbus (at the loss of some bandwidth between cages) or by using standard data communication links that disguise the underlying VMEbus architecture. There are no EMC (electromagnetic compatibility) requirements dictated by VMEbus, either conducted or radiated, nor are there power dissipation limits or chassis cooling requirements. VMEbus has left these issues to the system integrator, while VXIbus addresses these issues more rigorously.

Although electrically and logically similar to the 68000 microprocessor architecture, the VMEbus interface has been specified broadly enough that it is not dependent on any particular processor, and many processors are already supported on VMEbus, including the 80386. Many of the simpler VMEbus boards do not have processors at all.

A minimum VMEbus system requires only the P1 connector. All handshaking, arbitration, and interrupt support exists on P1, with P2 used to expand the system to 32 bits of address and data (A32 and D32). P1 will support 16 bit and 24 bit addressing (A16 and A24), as well as 8 and 16 bit data paths (D08 and D16). The extra lines needed for A32 and D32 are contained on the center row of P2, while the outer rows are user defined. These undefined pins are typically used for interface connections, such as allowing a module to drive a chassis mounted connector, access an internal disk drive, or provide for module to module communication. VSB (VMEbus Subsystem Bus) is a standard "subsystem bus" that has defined P2 as an additional communication path for up to six modules. Multiple VSBs may exist within any one VMEbus system. This is important to note, because VXIbus defines a subsystem of up to 13 modules and, like VSB, multiple VXIbus subsystems may exist within any one VXIbus system.

A.2.3 The VXIbus Extensions

VXIbus retains P1 and the center row of P2 exactly as defined by VMEbus. This includes the 5 volt and ±12 volt power pins on P1, and the additional 5 volt pins on P2. VXIbus includes the A and B card sizes, and these modules remain totally VMEbus compatible. However, VXIbus has made substantial additions to the VMEbus specification oriented towards instrumentation that can best be described as an electromechanical superset and a logical subset.

A.2.3.1 VXIbus MODULES

VXIbus has added two Eurocard module sizes of about 13 inch depth referred to as the C and D sizes. These modules are 9 and 14 inches high respectively, and are placed on 1.2 inch centers. The C Eurocard is the same height as the VMEbus B size board, and may sport both the P1 and P2 connectors. The D size module is a triple high Eurocard that may include a P3 connector in addition to P1 and P2. The 1.2 inch module width allows feasible implementation of high density instrumentation modules while allowing enough space for shielding both sides of a module and inserting an optional chassis shield. It also has the added benefit of allowing a high degree of compatibility with the shorter and narrower A and B sizes by allowing them to be mounted on full length board carriers or adapters. These carriers/adapters may also shield the sides of standard VMEbus cards, giving them a high degree of electromagnetic compatibility with VXIbus systems.

A.2.3.2 VXIbus SUBSYSTEMS

A VXIbus system may have up to 256 devices, including one or more VXIbus subsystems. A VXIbus subsystem consists of a central timing module referred to as Slot 0 with up to 12 additional instrument modules. P2 and P3 are completely defined in a VXIbus subsystem. These 13 modules conveniently fill a standard 19 inch cabinet when mounted vertically on 1.2 inch centers. Many VXIbus systems will consist only of a single frame with these 13 modules. A common configuration will load the Slot 0 module with system resources such as the VXIbus mandated timing generation, the VMEbus required system controller functions, and a data communication port such as IEEE 488 or RS-232. Slot 0 may also include optional instrumentation. The other positions are general purpose slots for the user to mix and match modules. A single VXIbus subsystem may have less than 12 additional slots, but may not have more. Any combination of VXIbus subsystems may exist within a VXIbus system. For instance, one VXIbus system may consist of a frame with one Slot 0 and 12 VXIbus modules extended to another frame that has a Slot 0 adjacent to three instrument slots, another Slot 0 with five instrument slots, and four standard VMEbus slots of undefined P2.

A.2.3.2.1 P2 Connector Definition

As mentioned previously, a VXIbus subsystem defines all P2 and P3 pins. The VXIbus P2 adds a 10 MHz ECL clock, ECL and analog supply voltages, ECL and TTL trigger lines, an analog summing bus, a module identification line, and a daisy chain structure known as the local bus. The trigger lines serve primarily as resources for signaling between instruments in a VXIbus subsystem, while the local bus lines are preferred for use within a multiple module instrument set (adjacent slots). The daisy chain local bus use is left to the module manufacturer to define, and several classes of electrical signals are permitted. Allowed signals are TTL, ECL, low voltage analog, and analog up to 42 volts. A keying mechanism near the faceplate indicating that module's local bus class prevents incompatible classes from accidentally being placed adjacently and potentially causing a destructive condition. Typical uses of the local bus include creating an internal analog bus or a chain of serial digital signal processors. There are a total of 24 local bus pins on P2, 12 lines in and 12 lines out for each slot; thus creating a 12 line bus that may or may not be passed on to adjacent slots.

A.2.3.2.2 P3 Connector Definition

The VXIbus P3 connector adds many of the same resource types as described for P2, but is aimed at higher performance instrumentation. Included on P3 is a 100 MHz clock and sync signal, additional power pins of the same supply voltages, more ECL trigger lines, and 24 additional lines (48 pins) of daisy chain local bus. Also defined on P3 is a "star" trigger system where precision ECL trigger signals are routed through Slot 0 acting as a cross point switch. This allows very precisely matched trigger timing between modules regardless of module position.

A.2.3.3 VXIbus SYSTEM ARCHITECTURE

The VXIbus device protocols define how modules are granted non-conflicting portions of the VMEbus address space. A device is typically a single module, but this is not required. Several devices may exist on a single module, and a single device may consist of multiple modules. 256 devices may exist in any one VXIbus system, and are referred to by logical device addresses ranging from 0 to 255. A VXIbus system configuration space is defined in the upper 16K of the 64K A16 address space. Each device is granted a total of 64 bytes in this space, which is sufficient for many of the simpler devices. Devices requiring additional address space have their address requirements readable in a defined register in the A16 address space. A "resource manager" reads this value shortly after power-on, and then assigns the requested memory space by writing the module's new VMEbus address into the device's offset register. This method positions a device's additional memory space in the A24 (16 Mbyte) or A32 (4 Gbyte) address space. If present day VMEbus cards are used in a system, the resource manager must position the VXIbus devices around the space taken by the standard VMEbus cards.

Higher level communication protocols are defined to allow sharing of interface modules and other devices by multiple manufacturers.

A.2.3.4 VXIbus SPECIFICATION STRUCTURE

In structure, the VXIbus specification parallels much of the VMEbus C.1 specification. Some VXIbus specification sections have the same titles as the VMEbus chapters. Other sections are devoted to topics not addressed by the VMEbus specification. The VXIbus document contains more than just the rules needed for VXIbus compliance; included are design tips and suggestions added to help the designer create compatible components. The same format of RULES, RECOMMENDATIONS, SUGGESTIONS, and OBSERVATIONS as VMEbus is used. As experience is gained creating VXIbus devices and using the specification, appropriate updates to the specification will occur.

A.3 DOCUMENT STRUCTURE

This document is organized in sections, with each section discussing a particular independent level of the implementation. In the event where a section parallels another standard document *(such as VMEbus)*, subsections will have corresponding numbers. Several appendices, containing clarifications and extensions, follow the body of the specification.

A.4 SPECIFICATION OBJECTIVES

This document defines a set of **RULES** and **RECOMMENDATIONS** for constructing a component which will interface to the VXIbus Family. The specifications cover the complete spectrum from basic hardware issues such as card dimensions to recommendations for communications protocols. This specification has the following objectives:

1. To allow communication among devices in a unambiguous fashion.
2. To allow for physical size reduction of standard *Rack&stack* instrumentation systems.
3. To allow for software cost reduction in test system integration by the use of common interfaces to analogous capabilities.
4. To provide higher system throughput for test systems through the use of higher bandwidth channels for inter-device communication, and the use of new protocols specifically designed to enhance throughput.
5. To provide test equipment which could be used in military IAC (Instrument on a Card) systems.
6. To provide the ability to implement new functionality in test systems through the use of virtual instruments.
7. To define how to implement Multi-module Instruments within the framework of this standard.

A.5 DEFINITION of TERMS

Throughout this document you will see the following headings on paragraphs. These headings identify the contents of the paragraph:

RULE: Rules **SHALL** be followed to ensure compatibility for cards in the system. A rule is characterized by the use of the words **SHALL** and **SHALL NOT**. These words are not used for any other purpose other than stating rules.

RECOMMENDATION: Recommendations consist of advice to implementors which will affect the usability of the final device. Discussions of particular hardware to enhance throughput would fall under a recommendation. These should be followed to avoid problems and to obtain optimum performance.

SUGGESTION: A suggestion contains advice which is helpful but not vital. The reader is encouraged to consider the advice before discarding it. Suggestions are included to help the novice designer with areas of design that can be problematic.

PERMISSION: Permissions are included to clarify the areas of the specification that are not specifically prohibited. Permissions reassure the reader that a certain approach is acceptable, and will cause no problems. The word **MAY** is reserved for indicating permissions.

OBSERVATION: Observations spell out implications of rules and bring attention to things that might otherwise be overlooked. They also give the rationale behind certain rules, so that the reader understands why the rule must be followed.

Any text which appears without heading should be considered as description of the standard.

B. VXIbus IMPLEMENTATION of VMEbus SPECS:

This document describes additions to and recommendations for the VMEbus specifications.[1] This section is organized in a parallel fashion to the VMEbus specification document. Sections B.1 - B.7 are additions to the corresponding VMEbus Revision C.1 Chapters 1-7, and Section B.8 is new material not contained in the VMEbus specifications.

B.1 INTRODUCTION

The intent of the VXIbus Implementation of the VMEbus specifications is to produce an electro-mechanical superset and logical subset of the VMEbus specifications in order to establish compatibility for all VXIbus systems. It is expected that all boards designed to the VMEbus specification will be compatible with this implementation mechanically and electrically. However, a subset of VMEbus Protocols relating to data transfer, arbitration, and interrupts has been specified, and may limit the utility of some standard VMEbus products in a VXIbus system. Mechanical extensions are defined to allow for a superset of board sizes.

B.2 DATA TRANSFER BUS

B.2.1 Address Modifiers

RECOMMENDATION B.2.1:
It is recommended that the address modifiers listed in the VMEbus specification as USER DEFINED not be used.

B.2.2 DTACK* Operation

RULE B.2.1:
A SLAVE **SHALL** pull DTACK* or BERR* low within 20 μs after a data strobe goes low.

RECOMMENDATION B.2.2:
A SLAVE should pull DTACK* low within 1 μs after a data strobe goes low.

RULE B.2.2:
A SLAVE **SHALL** release DTACK* and BERR* within 5 μs after the data strobe(s) goes high.

RECOMMENDATION B.2.3:
A SLAVE should release DTACK* and BERR* within 0.5 μs after the data strobe(s) goes high.

B.2.3 Bus Timer Operation

RULE B.2.3:
The BUS TIMER **SHALL** be BTO(≥100).

OBSERVATION B.2.1:
The BTO(100) capability minimum is to allow for chassis to chassis communication. There is a potential for each chassis to take up to 20 μs to respond to a data transfer.

B.3 DATA TRANSFER BUS ARBITRATION

RULE B.3.1:
The BUS GRANT daisy-chain lines **SHALL** be used or passed by all single boards and board assemblies.

RECOMMENDATION B.3.1:
The arbiter should implement priority based arbitration between the 4 Bus Request/Grant levels.

B.4 PRIORITY INTERRUPT

RULE B.4.1:
The INTERRUPT ACKNOWLEDGE daisy-chain lines **SHALL** be used or passed by all single boards and board assemblies.

B.5 UTILITIES

B.5.1 Power Monitor

The POWER MONITOR is a VMEbus function which asserts the SYSRESET* line at power-on, and asserts both the ACFAIL* and SYSRESET* lines immediately prior to power-removal. Although this is an optional VMEbus capability which may reside on the system controller board, it is a required VXIbus capability closely linked to the mainframe power supply. Therefore, each mainframe is required to provide this function.

RULE B.5.1:
Every VXIbus mainframe **SHALL** provide a VMEbus POWER MONITOR.

RECOMMENDATION B.5.1:
A power monitor should drive ACFAIL* low a minimum of 8 ms before driving SYSRESET* low, and a minimum of 10 ms before +5 Vdc power drops below 4.875 volts, when a power failure is detected. See Figure 5-4 of the VMEbus Specification, Rev. C.1.

B.5.2 Power Pins

See Section B.8, "EMC and System Power", for power pin specifications.

B.6 ELECTRICAL SPECIFICATIONS

RULE B.6.1:
All conductors on P1, P2, and P3 **SHALL** meet the requirements of Section 6.1 of the VMEbus Rev C.1 Electrical Specification.

B.6.1 The P1 Connector

The P1 connector shall remain as defined by the VMEbus specification (Rev C.1).

B.6.2 The VXIbus Subsystem P2 Connector

The VXIbus subsystem bus defines the outer rows (user defined pins) for instrumentation use. The center row retains the VMEbus specification. Alternate definitions of the outer two rows of P2 are permitted in VXIbus systems (not VXIbus subsystems) with a "Segmented Backplane" architecture where other VMEbus subsystem buses, such as VSB, may exist. A VXIbus subsystem bus consists of a system resource module defined as Slot 0 with up to 12 adjacent modules in increasing slot numbers. VXIbus and VXIbus subsystem bus are used interchangeably in this section of the VXIbus specification.

The VXIbus P2 connector delivers resources to modules particularly oriented toward instrumentation. On P2 it adds:

-5.2V, -2V, ±24V, and additional +5V power.

10 MHz differential clock.

2 parallel ECL trigger lines.

8 parallel TTL trigger lines.

Module identification pin.

12 lines of manufacturer defined local bus lines that connect to adjacent modules.

50 Ω terminated analog summing bus.

The Slot 0 module serves as a system resource and has a modified P2 functionality for central handling of the module identification pins. The Slot 0 Module, while providing these common resources, may also contain other devices and instrumentation.

B.6.2.1 CLK10

CLK10 is a 10 MHz system clock. It is sourced from Slot 0 and distributed to Slots 1-12 on P2. The Slot 0 output is differential ECL. It is buffered on the backplane and distributed to each module slot as a single source, single destination differential ECL signal. The CLK10 is individually buffered on the backplane for each slot position to provide a high level of inter-module isolation and relax the loading rules for modules. See Figure B.1.

RULE B.6.2:
The CLK10 frequency sourced from Slot 0 **SHALL** be 10 MHz. Its accuracy **SHALL** be equal to or better than ±100 ppm (.01%), over its specified operating temperature and time.

PIN NUMBER	ROWa SIGNAL MNEMONIC	ROWb SIGNAL MNEMONIC	ROWc SIGNAL MNEMONIC	PIN NUMBER
1	ECLTRG0	+5V	CLK10+	1
2	-2V	GND	CLK10-	2
3	ECLTRG1	RSV1	GND	3
4	GND	A24	-5.2V	4
5	LBUSA00	A25	LBUSC00	5
6	LBUSA01	A26	LBUSC01	6
7	-5.2V	A27	GND	7
8	LBUSA02	A28	LBUSC02	8
9	LBUSA03	A29	LBUSC03	9
10	GND	A30	GND	10
11	LBUSA04	A31	LBUSC04	11
12	LBUSA05	GND	LBUSC05	12
13	-5.2V	+5V	-2V	13
14	LBUSA06	D16	LBUSC06	14
15	LBUSA07	D17	LBUSC07	15
16	GND	D18	GND	16
17	LBUSA08	D19	LBUSC08	17
18	LBUSA09	D20	LBUSC09	18
19	-5.2V	D21	-5.2V	19
20	LBUSA10	D22	LBUSC10	20
21	LBUSA11	D23	LBUSC11	21
22	GND	GND	GND	22
23	TTLTRG0*	D24	TTLTRG1*	23
24	TTLTRG2*	D25	TTLTRG3*	24
25	+5V	D26	GND	25
26	TTLTRG4*	D27	TTLTRG5*	26
27	TTLTRG6*	D28	TTLTRG7*	27
28	GND	D29	GND	28
29	RSV2	D30	RSV3	29
30	MODID	D31	GND	30
31	GND	GND	+24V	31
32	SUMBUS	+5V	-24V	32

TABLE B.1. P2 Pin Definitions: Slots 1-12.

RECOMMENDATION B.6.1:

A Slot 0 module should allow CLK10 to be derived from an external frequency source.

OBSERVATION B.6.1:

An external frequency source would permit the use of an accurate frequency reference, such as a Rubidium Standard, and would facilitate synchronization of multiple VXIbus mainframes.

PIN NUMBER	ROWa SIGNAL MNEMONIC	ROWb SIGNAL MNEMONIC	ROWc SIGNAL MNEMONIC	PIN NUMBER
1	ECLTRG0	+5V	CLK10+	1
2	-2V	GND	CLK10-	2
3	ECLTRG1	RSV1	GND	3
4	GND	A24	-5.2V	4
5	MODID12	A25	LBUSC00	5
6	MODID11	A26	LBUSC01	6
7	-5.2V	A27	GND	7
8	MODID10	A28	LBUSC02	8
9	MODID09	A29	LBUSC03	9
10	GND	A30	GND	10
11	MODID08	A31	LBUSC04	11
12	MODID07	GND	LBUSC05	12
13	-5.2V	+5V	-2V	13
14	MODID06	D16	LBUSC06	14
15	MODID05	D17	LBUSC07	15
16	GND	D18	GND	16
17	MODID04	D19	LBUSC08	17
18	MODID03	D20	LBUSC09	18
19	-5.2V	D21	-5.2V	19
20	MODID02	D22	LBUSC10	20
21	MODID01	D23	LBUSC11	21
22	GND	GND	GND	22
23	TTLTRG0*	D24	TTLTRG1*	23
24	TTLTRG2*	D25	TTLTRG3*	24
25	+5V	D26	GND	25
26	TTLTRG4*	D27	TTLTRG5*	26
27	TTLTRG6*	D28	TTLTRG7*	27
28	GND	D29	GND	28
29	RSV2	D30	RSV3	29
30	MODID00	D31	GND	30
31	GND	GND	+24V	31
32	SUMBUS	+5V	-24V	32

TABLE B.2. P2 Pin Definitions: Slot 0.

RULE B.6.3:

CLK10 duty cycle **SHALL** be 50% ±5% when measured at the 50% transition level.

RULE B.6.4:

IF CLK10 is switched between different clock sources,
THEN the minimum pulse width, high or low, **SHALL NOT** be less than 30 ns or more than 10 μs during switching. The minimum time between two successive transitions of the same polarity **SHALL NOT** be less than 80 ns.

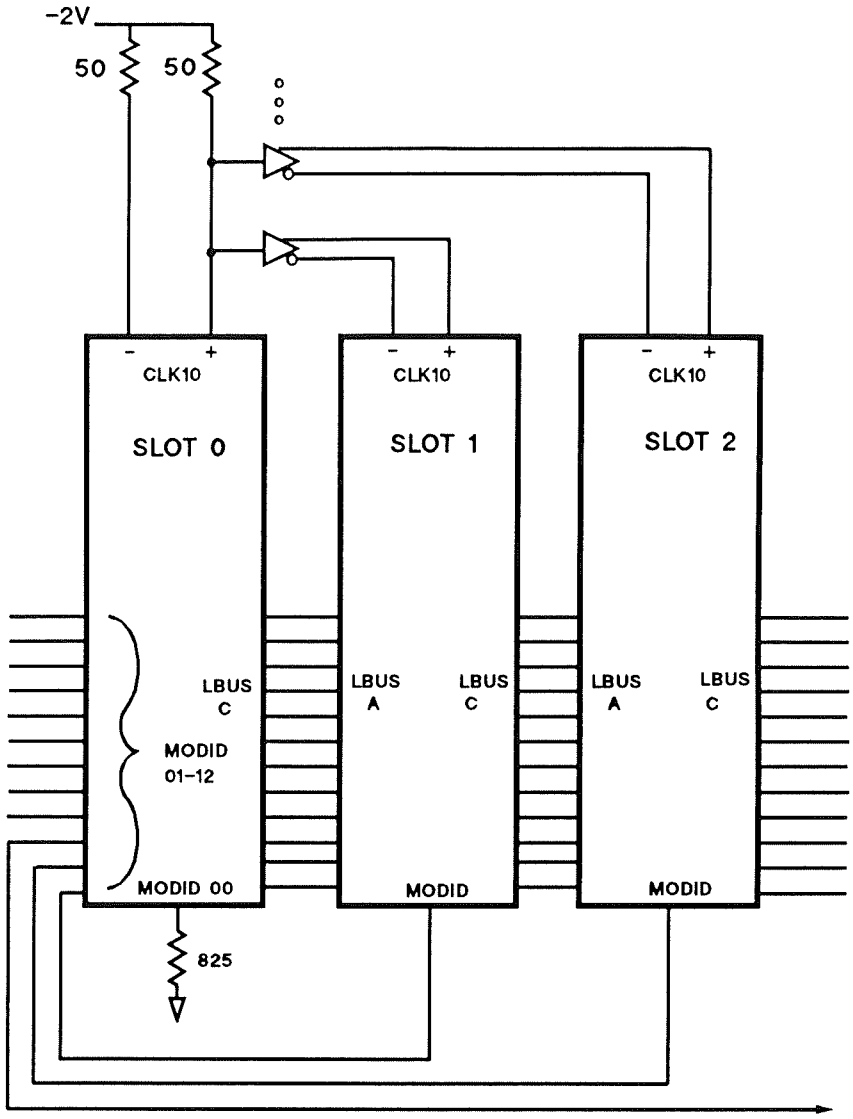


Figure B.1. CLK10, MODID, and LBUS backplane signal routing.

RULE B.6.5:
Each slot's CLK10 **SHALL** be driven by a unique backplane buffer output.

RULE B.6.6:
CLK10 **SHALL** be distributed differentially from Slot 0.

RULE B.6.7:
The backplane CLK10 distribution traces **SHALL** be designed for 50 Ω.

RULE B.6.8:
IF a module accesses the CLK10 signals,
THEN it **SHALL** provide 50 Ω termination on CLK10+ and CLK10-, with no more than two (2) equivalent ECL loads.

RULE B.6.9:
The absolute delay of CLK10 from Slot 0 to any module **SHALL NOT** exceed 8 ns.

OBSERVATION B.6.2:
Either single ended input buffers or differential input buffers can be used to buffer and fan out CLK10 on the backplane. Typical components are the 10H101 and 10H116 buffers.

B.6.2.2 MODID LINES

The MODID lines allow a logical device to be identified with a particular physical location or slot. These lines are sourced from the VXIbus Slot 0 module to the Slot 1 and higher modules. There is one line to each module, located on P2 pin A30. A total of 12 MODID lines will connect between Slot 0 and the other modules in a maximally configured VXIbus subsystem. In addition to these 12 lines, Slot 0 has its own MODID line (MODID00). The purpose of the MODID lines is to:

1. Detect the presence of a module in a slot, even if that module has failed.
2. Identify the geographical location (slot number) of a particular device.
3. Indicate by lamp or other means the actual physical location of the module.

Slot 0 detects the presence of a module by the module pulling down its MODID line with a fixed resistor to ground. This allows any module, even if failed and unpowered, to be detected.

The slot number of a device is identified by the Slot 0 module asserting a particular MODID line, and polling the MODID bit in each module's A16 configuration space to detect the selected device.

A visual indicator, such as a lamp, may be placed adjacent to a slot (or even on the module) to illuminate when that particular MODID line is driven true. This may be used to quickly identify the location of any module, including failed modules. Refer to Figure B.1 for rules for loading and driving the MODID lines.

RULE B.6.10:
Each Slot 0 MODIDxx receiver **SHALL** comply with the VMEbus loading rules for High Current 3 State lines (VMEbus Revision C.1, Rule 6.14).

RULE B.6.11:
Each Slot 0 MODIDxx driver **SHALL** comply with the VMEbus driving rules for Standard 3 State lines (VMEbus Revision C.1, Rule 6.15).

RULE B.6.12:
Each Slot 0 device **SHALL** provide a 16.9 KΩ pull up resistor from each MODID line to +5 volts.

RULE B.6.13:
IF a non-Slot 0 module accesses the MODID pin,
THEN it **SHALL** provide an 825 Ω pull down to ground in parallel with no more than 100 μA of leakage current on its MODID line.

OBSERVATION B.6.3:
16.9 KΩ and 825 Ω are standard 1% metal film resistor values.

RULE B.6.14:
A VXIbus subsystem backplane **SHALL** provide an 825 Ω pull down to ground on the MODID00 line.

RULE B.6.15:
The backplane **SHALL NOT** sink or source more than 100 μA of leakage current to any MODID line.

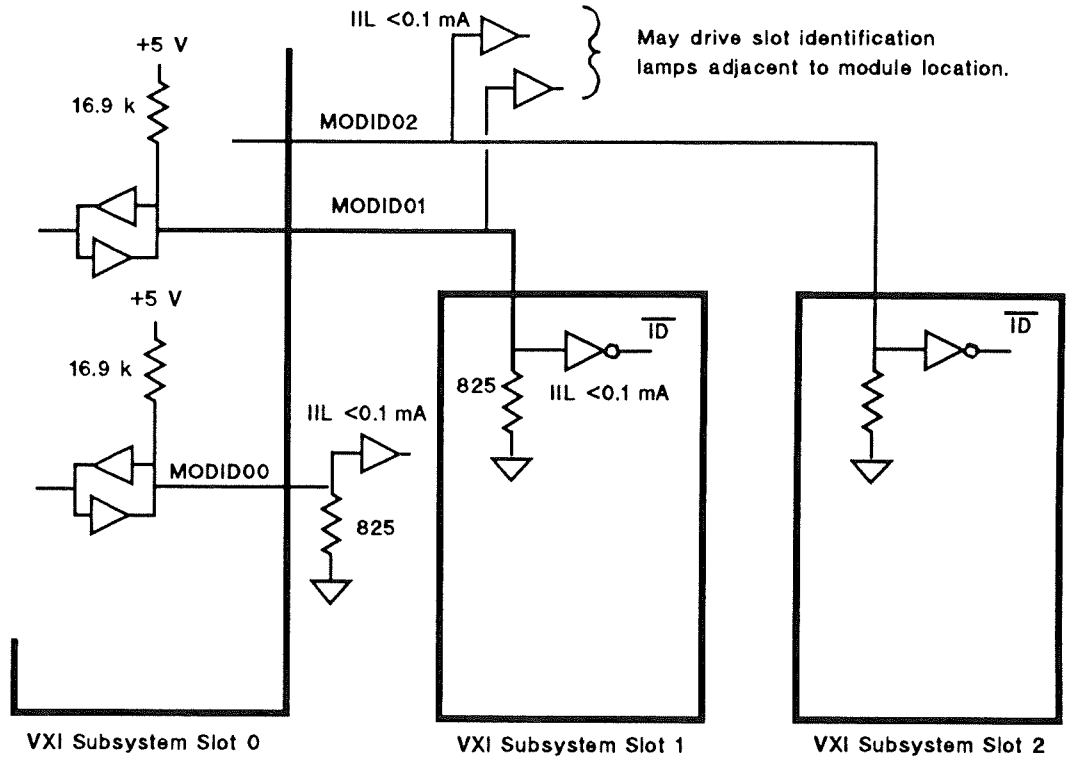


Figure B.2. Module ID lines.

PERMISSION B.6.1:

A VXIbus subsystem Slot 0 module **MAY** drive its MODID line true to illuminate its own module location lamp.

OBSERVATION B.6.4:

The proper placement of a Slot 0 module in a Slot 0 position may be verified by a read of its MODID00 state when all Slot 0 MODID drivers are disabled. A low state indicates that the module is in a Slot 0 position. A high state indicates that it is in another position.

B.6.2.3 TTLTRG0-7* (TTL TRIGGER LINES)

The TTLTRG* lines are open collector TTL lines used for intermodule communication. Any module, including Slot 0, may drive these lines and receive information on these lines. They are general purpose lines that may be used for trigger, handshake, clock, or logic state transmission. They are in a non-asserted (high) state until allocated by the user under program control. Some standard allocation procedures are defined. Standard protocols, synchronous (SYNC), asynchronous (ASYNC), and start/stop (STST) are defined. To compensate for the larger rise time caused by the passive pullup termination, these protocols specify separate timing requirements for trigger sources and trigger acceptors. When used for logic state transmission, setup and hold times with respect to a clock edge are defined. Other protocols may be defined by a manufacturer.

OBSERVATION B.6.5:
The VMEbus Specification Revision C.1 recommends a 100 Ω system impedance for open collector lines. Although the TTLTRG* lines will be terminated and meet the drive and load requirements for the VMEbus, the characteristic impedance of the traces on the backplane may be closer to 50 Ω. This allows ECL traces to share the same backplane layer, and may keep backplanes at a manufacturable number of planes and thickness for component insertion. This should not significantly affect performance of the TTLTRG* lines.

RULE B.6.16:
The backplane **SHALL** terminate all TTLTRG* lines as specified by the VMEbus specification, Rev. C.1, Figure 6-2, Standard Bus Termination.

RULE B.6.17:
A module's interface to any TTLTRG* signal line **SHALL** conform to the driving and loading rules for open collector lines (VMEbus Specification Revision C.1, Section 6.4.2.5).

RULE B.6.18:
All TTLTRG* line drivers **SHALL** be unasserted within one (1) second after SYSRESET* becomes unasserted.

RULE B.6.19:
A module **SHALL** allocate any TTLTRG* lines to specific functions in groups of 1, 2 or 4.

RULE B.6.20:
IF a module uses single line TTLTRG* groups,
THEN that module **SHALL** be user-programmable to connect to any TTLTRG* line.

RULE B.6.21:
IF a module uses two line TTLTRG* groups,
THEN that module **SHALL** be user programmable to connect all defined TTLTRG* group pairs. The defined TTLTRG* group pairs are lines 0 and 1, lines 2 and 3, lines 4 and 5, and lines 6 and 7.

RULE B.6.22:
IF a module uses four line TTLTRG* groups,
THEN that module **SHALL** be user-programmable to connect to all defined TTLTRG* four line groups. The defined TTLTRG* four line group are lines 0, 1, 2, and 3; and lines 4, 5, 6, and 7.

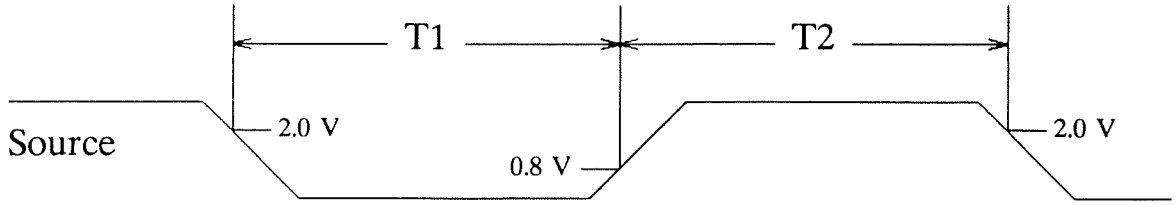
B.6.2.3.1 Standard TTLTRG* Protocols

B.6.2.3.1.1 TTLTRG* SYNCHRONOUS (SYNC) TRIGGER PROTOCOL.

The TTLTRG* SYNC trigger protocol is a single line broadcast trigger that does not require an acknowledge from any acceptors.

RULE B.6.23:
A TTLTRG* SYNC trigger source **SHALL** assert the trigger for a minimum time of T1 and **SHALL NOT** reassert the trigger for a minimum time of T2 as shown in Figure B.3.

RULE B.6.24:
A TTLTRG* SYNC trigger acceptor **SHALL** accept any trigger asserted for ≥10 ns, following a ≥10 ns deassertion. *



Parameter Label	Min	Max	Description
T1	30 ns		Minimum source assertion time
T2	50 ns		Minimum source non-assertion time

Figure B.3. TTLTRG* Synchronous (SYNC) Trigger Protocol.

B.6.2.3.1.2 TTLTRG* ASYNCHRONOUS (ASYNC) TRIGGER PROTOCOL

The TTLTRG* ASYNC trigger protocol is a two line single source, single acceptor protocol. The source initiates an operation by asserting the lower numbered line of the allocated TTLTRG* line pair, while the acceptor acknowledges by asserting the higher numbered line of the TTLTRG* line pair. This is a useful trigger mode for handshaking between VXIbus modules and external instruments or between VXIbus mainframes.

RULE B.6.25:
A module implementing the TTLTRG* ASYNC trigger protocol **SHALL** source only on the lower numbered line of a TTLTRG* line pair.

RULE B.6.26:
A module implementing the TTLTRG* ASYNC trigger protocol **SHALL** acknowledge only on the higher numbered line of a TTLTRG* line pair.

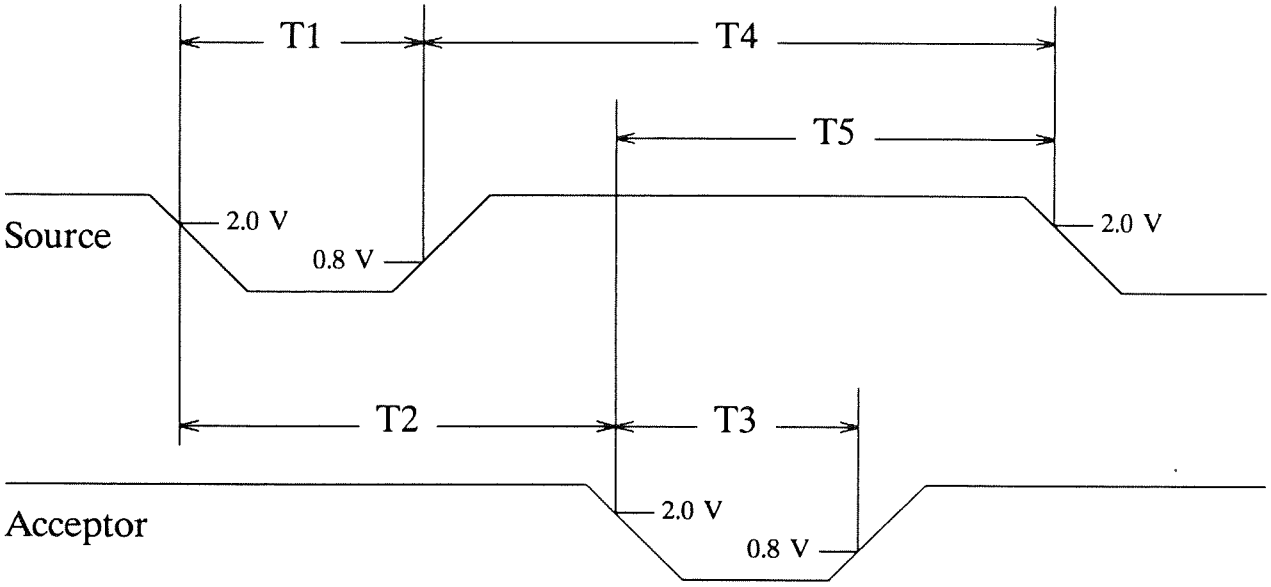
RULE B.6.27:
IF a module implements the TTLTRG* ASYNC trigger protocol
THEN it **SHALL** meet the timing requirements shown in Figure B.4.

RULE B.6.28:
A TTLTRG* ASYNC trigger source or acceptor **SHALL** accept any trigger asserted for $\geq$10 ns following a $\geq$10 ns deassertion.

B.6.2.3.1.3 TTLTRG* CLOCK TRANSMISSION

A TTLTRG* line may be used for a clock signal transmission.

RULE B.6.29:
IF A TTLTRG* line is used for clock transmission,
THEN it **SHALL** meet the timing requirements described for the TTLTRG* SYNC trigger protocol.



Parameter Label	Min	Max	Description
T1	30 ns		Minimum Source assertion time
T2	0 ns		Minimum Acceptor response time
T3	30 ns		Minimum Acceptor assertion time
T4	50 ns		Minimum Source non-assertion time
T5	0 ns		Minimum Acceptor Acknowledge to source re-assertion time

Figure B.4. TTLTRG* Asynchronous (ASYNC) Trigger Protocol.

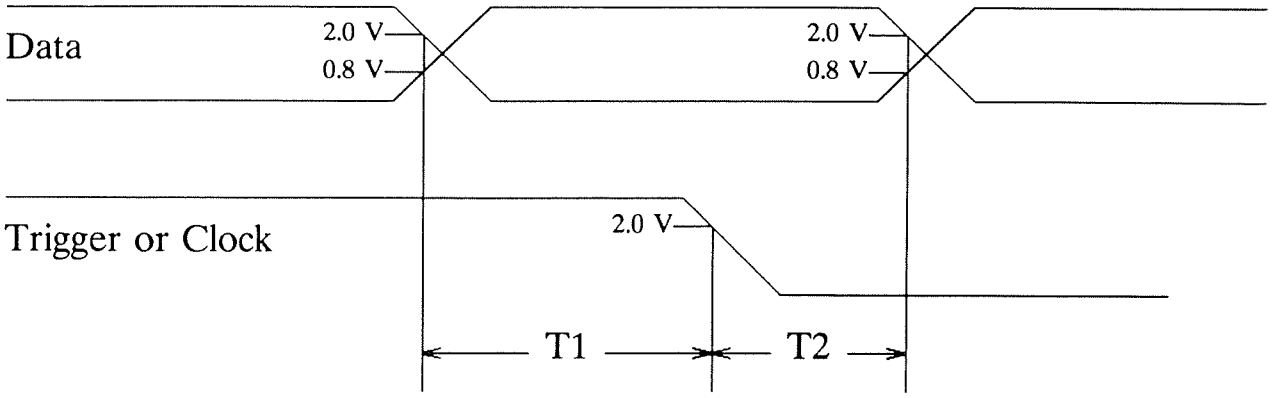
OBSERVATION B.6.6:
In a fully loaded VXIbus subsystem, the TTLTRG* rise time will approach 40 nanoseconds. For this reason, the falling edge is used for triggering.

B.6.2.3.1.4 TTLTRG* DATA TRANSMISSION

The TTLTRG* lines can also be used for data transmission, where one of the TTLTRG* lines is used as a clock. Data may be synchronized to either the rising or falling edge of the clock, or both.

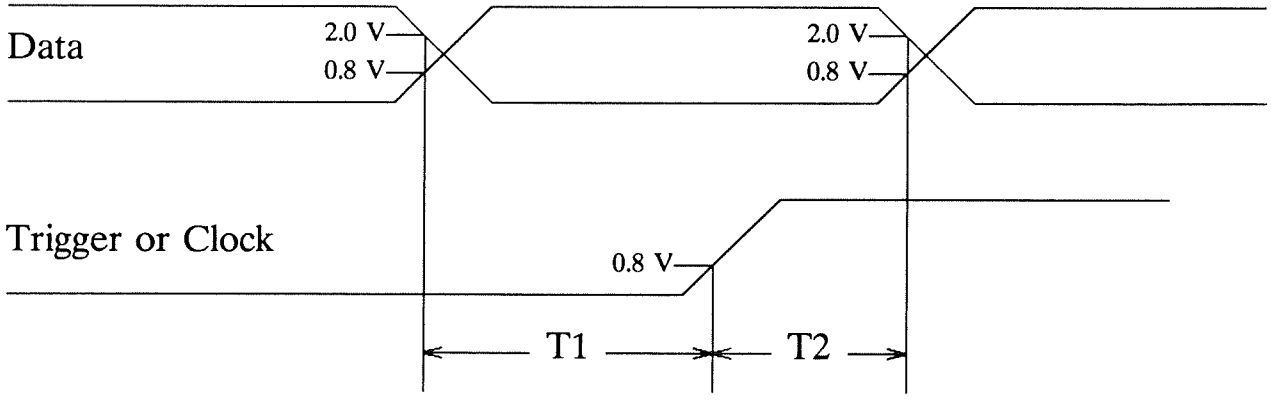
RULE B.6.30:
IF a source uses TTLTRG* lines for data transmission on a falling clock edge,
THEN the source **SHALL** meet the timing requirements shown in Figure B.5.

RULE B.6.31:
IF a source uses TTLTRG* lines for data transmission on a rising clock edge,
THEN the source **SHALL** meet the timing requirements shown in Figure B.6.



Parameter Label	Min	Max	Description
T1	40 ns		Data setup time
T2	10 ns		Data hold time

Figure B.5. TTLTRG* Data Transmission on falling clock edge.



Parameter Label	Min	Max	Description
T1	40 ns		Data setup time
T2	40 ns		Data hold time

Figure B.6. TTLTRG* Data Transmission on rising clock edge.

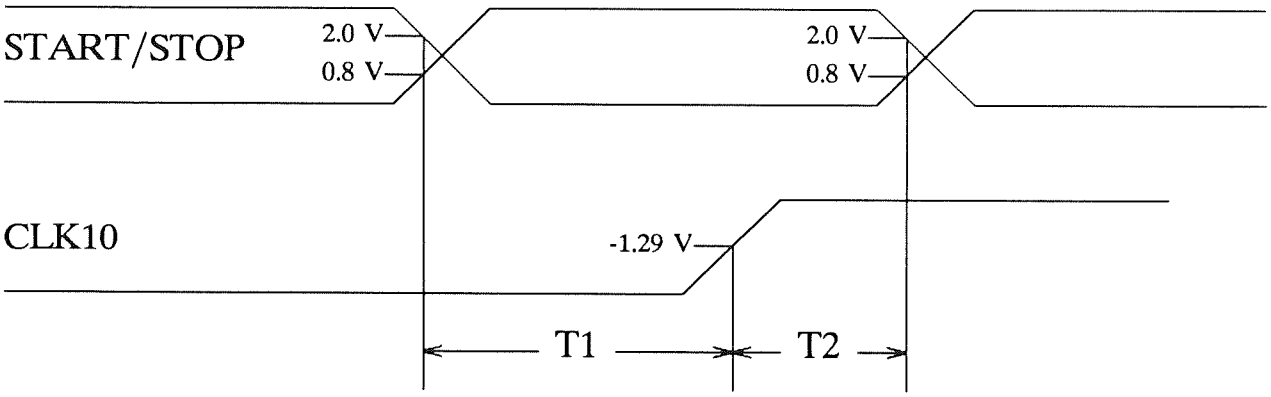
RULE B.6.32:
A TTLTRG* data acceptor **SHALL** accept any data having data setup and hold times of ≥7 ns each.

B.6.2.3.1.5 START/STOP (STST) PROTOCOL

The Start/Stop protocol (STST) provides a method for starting and stopping module clusters synchronously. One TTLTRG* line is driven from the Slot 0 module and its state signifies START or STOP operation. All participating modules respond to this line synchronously at the next CLK10 rising edge. Slot 0 has the responsibility to synchronize the START/STOP signal with CLK10 so that setup and hold times are guaranteed for all acceptors. This may require an arbiter in Slot 0 to synchronize external events.

RULE B.6.33:
IF a module implements STST protocol,
THEN the START/STOP line **SHALL** be user-programmable to connect to any TTLTRG* line.

RULE B.6.34:
IF a Slot 0 module implements the STST protocol,
THEN it **SHALL** meet the timing requirements shown in Figure B.7.



Parameter Label	Min	Max	Description
T1	50 ns		START/STOP setup time
T2	15 ns		START/STOP hold time

Figure B.7. TTLTRG* START/STOP timing.

RULE B.6.35:
A TTLTRG* STST acceptor **SHALL** accept any STST command having data setup and hold times of ≥7 ns each.

RULE B.6.36:
When implementing the STST protocol, START **SHALL** be represented by a low (asserted) state on the START/STOP TTLTRG* line, and STOP **SHALL** be represented by a high (unasserted) state on the START/STOP TTLTRG* line.

B.6.2.3.1.6 EXTERNAL TRIGGER BUFFERING

OBSERVATION B.6.7:

The TTLTRG* loading and driving rules require that these lines be buffered when extended outside a mainframe.

RECOMMENDATION B.6.2:

When extending a trigger operation external to a VXIbus chassis, the buffer module should source and accept signals as described in the Standard External Trigger Buffer below.

B.6.2.3.1.6.1 Standard External Trigger Buffer

To maximize the compatibility between multiple VXIbus mainframes or a VXIbus mainframe and external instruments, a standard buffering scheme for TTLTRG* lines is recommended. A source has an output series impedance of 50 Ω. Its open circuit low level output is ≤0.4 V, and its open circuit high level output is ≥4.2 V. 50 Ω cable is recommended for connection from the source module to other VXIbus mainframes or instruments. For highest performance, terminate the cable with 50 Ω at the furthest point from the source. The acceptor's threshold level is centered at 1.5 V with a loading impedance of ≥5 kΩ. The TTLTRG* sense remains uninverted (Asserted state is low). See Figure B.8. *

OBSERVATION B.6.8:

The Standard External Trigger Buffer requires only simple buffering to implement the SYNC and ASYNC trigger protocols. *

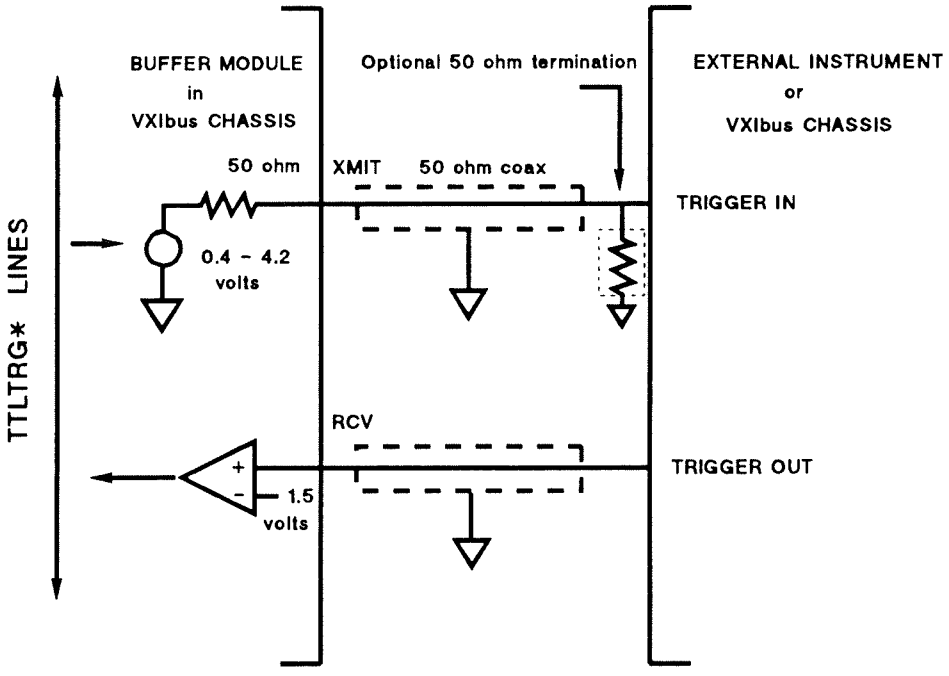


Figure B.8. External Trigger Buffering

B.6.2.4 ECLTRG0-1 (ECL TRIGGER LINES)

The ECLTRG lines are intended to be an intermodule timing resource. These two lines are bussed the length of the VXIbus subsystem backplane. Any module, including the Slot 0 module, may drive these lines and receive information from these lines. These lines are single-ended ECL, and have a system impedance of 50 Ω. The asserted state is defined as logical high.

RULE B.6.37:
Each ECLTRG line **SHALL** be terminated with 50 Ω to -2 V at each end of the backplane. This requires that any driver be able to drive 25 Ω.

RULE B.6.38:
The maximum ECLTRG trace length from the P2 DIN connector pad on any module **SHALL NOT** exceed 1.0 inch.

RECOMMENDATION B.6.3:
The transmission line impedance connecting the P2 DIN connector to the ECLTRG receiver/driver circuitry should be 50 Ω.

RECOMMENDATION B.6.4:
A module that senses the ECLTRG lines should have a 150 Ω resistor in series with the transmission line connecting the ECL trigger lines on the P2 DIN connector to the line receiver.

RULE B.6.39:
Each module **SHALL NOT** have more than one equivalent driver/receiver load on any ECLTRG line. *

RULE B.6.40:
IF a module drives the ECLTRG bus
THEN it **SHALL** use ECL drivers which provide a LOW output level which is less (more negative) than the -2V termination supply, e.g. 10H123.

OBSERVATION B.6.9:
Using drivers which produce a high impedance LOW level for ECLTRG lines eliminates the glitching which occurs when ECL devices are WIRE-OR'ed.

RULE B.6.41:
IF a module connects to a ECLTRG line
THEN it **SHALL** unassert that within one (1) second after SYSRESET* becomes unasserted.

RULE B.6.42:
ECLTRG lines **SHALL** be allocated in groups of 1 or 2 and meet the same allocation rules as defined for the TTLTRG* lines.

Figure B.9 shows a typical ECL interface.

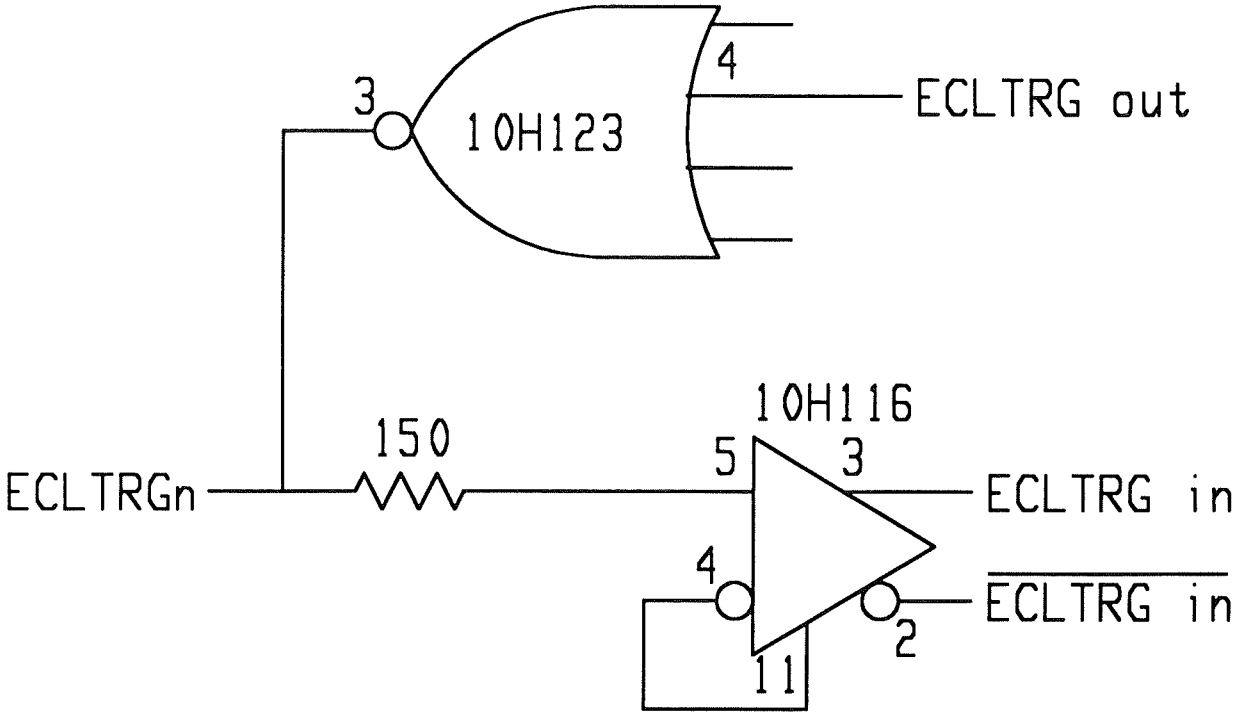


Figure B.9. Typical ECLTRG Interface.

B.6.2.4.1 Standard ECLTRG Protocols

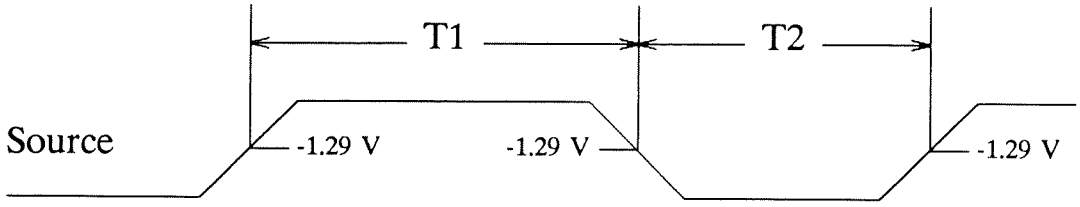
The ECLTRG lines have a set of defined protocols corresponding to the TTLTRG* protocols. These protocols are defined in the following sections.

B.6.2.4.1.1 ECLTRG SYNCHRONOUS (SYNC) TRIGGER PROTOCOL.

The ECLTRG SYNC trigger protocol is a single line broadcast trigger that does not require an acknowledge from any acceptors.

RULE B.6.43:
A ECLTRG SYNC trigger source **SHALL** assert the trigger for a minimum time of T1 and **SHALL NOT** reassert the trigger for a minimum time of T2 as shown in Figure B.10.

RULE B.6.44:
An ECLTRG SYNC trigger acceptor **SHALL** accept any trigger asserted for ≥6 ns following a ≥6 ns deassertion. *



Parameter Label	Min	Max	Description
T1	8 ns		Minimum source assertion time
T2	8 ns		Minimum source non-assertion time

Figure B.10. ECLTRG Synchronous (SYNC) Trigger Protocol.

B.6.2.4.1.2 ECLTRG ASYNCHRONOUS (ASYNC) TRIGGER PROTOCOL

The ECLTRG ASYNC trigger protocol is a two line single source, single acceptor protocol. The source initiates an operation by asserting the lower numbered line of the allocated ECLTRG line pair, while the acceptor acknowledges by asserting the higher numbered line of the ECLTRG line pair. This is a useful trigger mode for handshaking between VXIbus modules and external instruments.

RULE B.6.45:
A module implementing the ECLTRG ASYNC trigger protocol **SHALL** source only on the lower numbered line of a ECLTRG line pair.

RULE B.6.46:
A module implementing the ECLTRG ASYNC trigger protocol **SHALL** acknowledge only on the higher numbered line of a ECLTRG line pair.

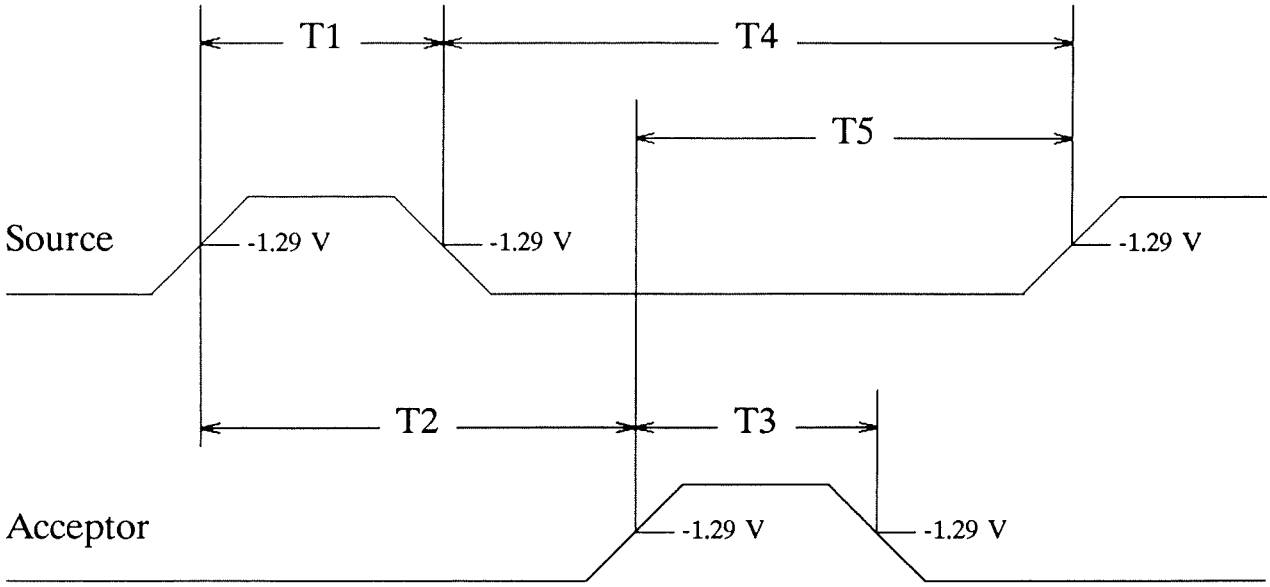
RULE B.6.47:
IF a module implements the ECLTRG ASYNC trigger protocol
THEN it **SHALL** meet the timing requirements shown in Figure B.11.

RULE B.6.48:
An ECLTRG ASYNC trigger source or acceptor **SHALL** accept any trigger asserted for $\geq$6 ns following a $\geq$6 ns deassertion.

B.6.2.4.1.3 ECLTRG CLOCK TRANSMISSION

A ECLTRG line may be used for a clock signal transmission.

RULE B.6.49:
IF A ECLTRG line is used for clock transmission,
THEN it **SHALL** meet the timing requirements described for the ECLTRG SYNC trigger protocol.



Parameter Label	Min	Max	Description
T1	8 ns		Minimum Source assertion time
T2	0 ns		Minimum Acceptor response time
T3	8 ns		Minimum Acceptor assertion time
T4	8 ns		Minimum Source non-assertion time
T5	0 ns		Minimum Acceptor Acknowledge to source re-assertion time

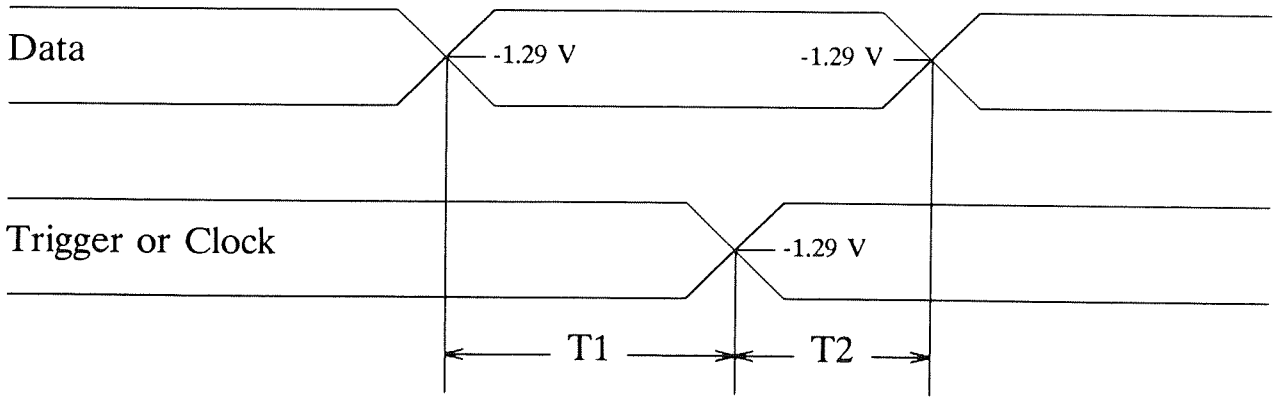
Figure B.11. ECLTRG Asynchronous (ASYNC) Trigger Protocol.

B.6.2.4.1.4 ECLTRG DATA TRANSMISSION

The ECLTRG lines can also be used for data transmission, where one of the ECLTRG lines is used as a clock. Data may be synchronized to either the rising or falling edge of the clock, or both.

RULE B.6.50:
IF a source uses ECLTRG lines for data transmission,
THEN the source **SHALL** meet the timing requirements shown in Figure B.12.

RULE B.6.51:
An ECLTRG data acceptor **SHALL** accept any data with a setup time of ≥6 ns and a hold time of ≥2 ns.



Parameter Label	Min	Max	Description
T1	8 ns		Data setup time
T2	4 ns		Data hold time

Figure B.12. ECLTRG Data Transmission.

B.6.2.4.1.5 EXTENDED START/STOP (ESTST) PROTOCOL

Extended START/STOP (ESTST) protocol is the natural extension of the START/STOP (STST) protocol to D size modules. Because the modules may contain P3 and utilize CLK100 as their instrument clock, three additional signals have been added to the STST protocol (CLK100, SYNC100, and an ECLTRG line). The ECLTRG line determines whether an ESTST operation is initiated, while the same TTLTRG line used to transmit the START/STOP value for STST devices also determines the START or STOP operation for ESTST devices.

In the ESTST protocol, a Slot 0 SYNC100 driver (See Sec. B.6.3.2, "SYNC100") selects a particular CLK100 clock edge to initiate the START or STOP command. An ECLTRG line indicates whether the SYNC100 pulse is related to the ESTST protocol. This line is known as the ESTST qualifier line. Since the CLK10 and CLK100 signals are synchronized, the Slot 0 module can always qualify a CLK100 edge having a fixed phase relationship to CLK10. This fixed phase relationship, over a series of ESTST operations, guarantees fixed and repeatable delays between modules paced by CLK10 (STST) and those paced by CLK100 (ESTST). The TTLTRG* line used to indicate START/STOP for the STST protocol also determines START or STOP for the ESTST protocol.

RULE B.6.52:
IF a module implements the ESTST protocol,
THEN the module **SHALL** be user programmable to select any ECLTRG line as the ESTST qualifier line.

RULE B.6.53:
ESTST qualifier true **SHALL** be represented by a high state of the selected ECLTRG ESTST qualifier line, and ESTST qualifier false **SHALL** be represented by a low state.

RULE B.6.54:
A Slot 0 module implementing the ESTST protocol **SHALL** meet the timing requirements of Figure B.13.

RULE B.6.55:
A Slot 0 module implementing the ESTST protocol **SHALL** meet the timing requirements for the STST protocol.

RULE B.6.56:
An ESTST acceptor **SHALL** accept a SYNC100 setup value for $\geq$2 ns and held for $\geq$0.5 ns, an ESTST qualifier value setup for $\geq$4 ns and held for $\geq$2 ns, and a STST value setup $\geq$7 ns and held for $\geq$7 ns with respect to the CLK100 edge.

B.6.2.4.1.6 EXTERNAL TRIGGER BUFFERING

RECOMMENDATION B.6.5:
Since the ECLTRG asserted sense is opposite the TTLTRG* asserted sense, the ECLTRG signal should be inverted by the Standard External Trigger Buffer. This allows compatibility of external ECLTRG and TTLTRG* signals.

OBSERVATION B.6.10:
By meeting the loading and driving rules for ECLTRG lines, a module is prohibited from extending these lines external to the chassis without buffering these lines. See Section B.6.2.3.1.6.1, "Standard External Trigger Buffer." *

B.6.2.5 SUMBUS

The SUMBUS is an analog summing node that is bussed the length of the VXIbus subsystem backplane. It is intended that any module be able to drive or receive from this line. Each module can drive this line using an analog current source driver. Each module may receive information from this bus through a high impedance receiver, such as a high impedance analog buffer amplifier.

RULE B.6.57:
The SUMBUS **SHALL** be terminated to signal ground through 50 $\pm$1 Ω at each end of the backplane.

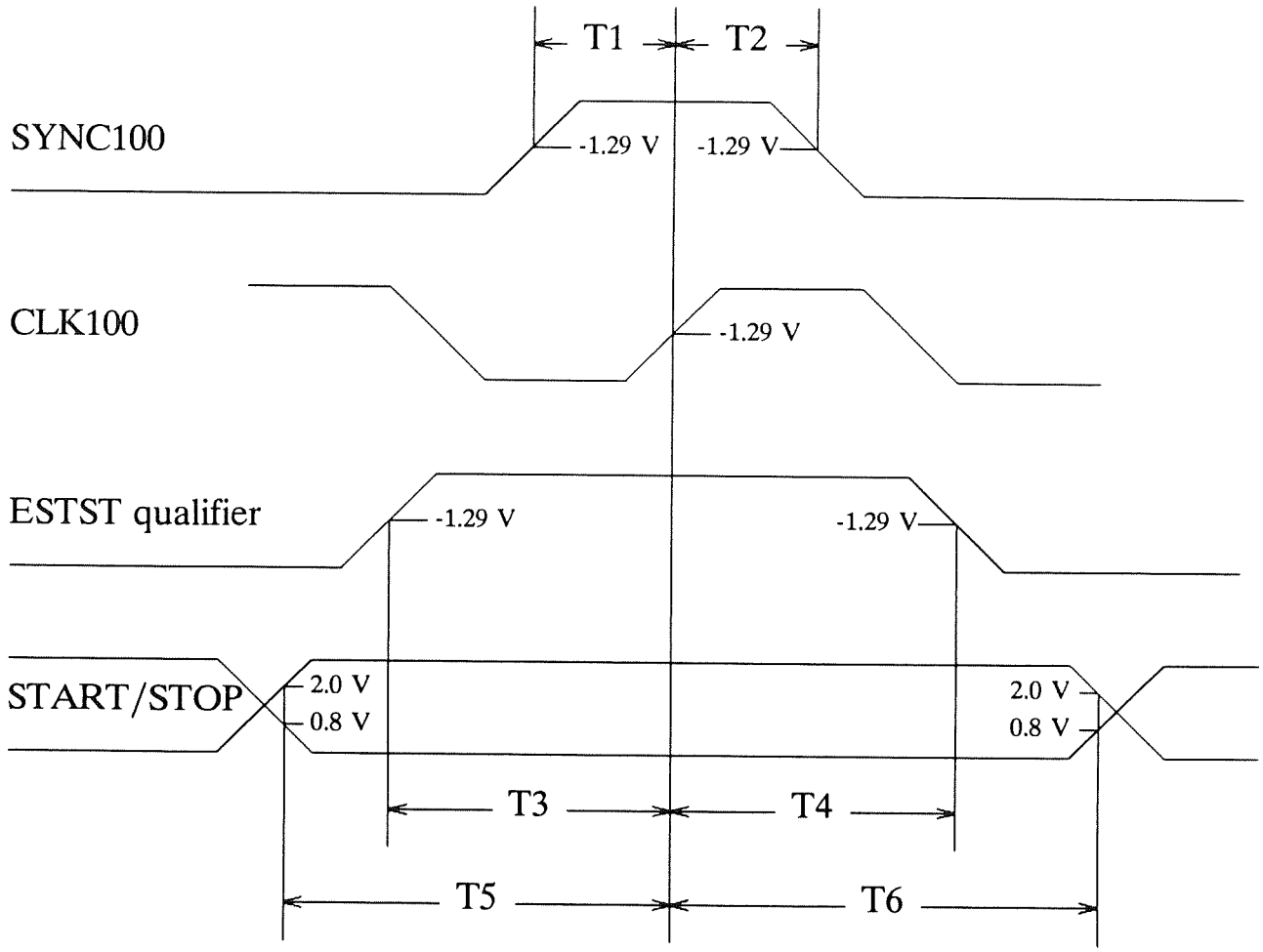
RULE B.6.58:
A receiver on the SUMBUS **SHALL** have an equivalent input impedance of $\geq$10 KΩ in parallel with $\leq$3 pF.

RULE B.6.59:
A SUMBUS output current source **SHALL** have an equivalent output impedance of $\geq$10 KΩ in parallel with $\leq$4 pF.

RULE B.6.60:
A SUMBUS output current source **SHALL** have $\pm$0.8 V minimum compliance.

RULE B.6.61:
A SUMBUS output current source **SHALL NOT** source more than 40 mA.

RULE B.6.62:
The backplane **SHALL** supply a SUMBUS protection network that clamps the SUMBUS voltage within the range $\pm$3.0 V for all SUMBUS input currents in the range of $\pm$520 mA.



Parameter Label	Min	Max	Description
T1	4.0 ns	7.5 ns	SYNC100 setup time
T2	3.5 ns	6.0 ns	SYNC100 hold time
T3	13.0 ns		ESTST Qualifier setup time
T4	11.0 ns		ESTST Qualifier hold time
T5	50.0 ns		START/STOP setup time
T6	15.0 ns		START/STOP hold time

Figure B.13. CLK100, SYNC100, and ECLTRG START/STOP Timing.

RULE B.6.63:
The SUMBUS protection network **SHALL NOT** source or sink more than ±1.0 μA nor load with greater than 30 pF for SUMBUS voltages in the range ±1.0 V.

OBSERVATION B.6.11:
Low leakage diodes may be required to meet the SUMBUS leakage requirement.

SUGGESTION B.6.1:
Because all modules may drive the SUMBUS at the same time, a module designer should consider the total dynamic range of a SUMBUS output current source and be able to adjust the maximum output current accordingly. This will keep the signal within the compliance region.

RULE B.6.64:
SUMBUS output current sources and receivers **SHALL NOT** be located more than 1.0 inch from the P2 DIN SUMBUS pad.

RULE B.6.65:
When a module's SUMBUS module output current source is disabled, the module **SHALL NOT** source more than ±1.0 μA into the SUMBUS.

B.6.2.6 LOCAL BUS

The Local Bus is a daisy chained bus. It is defined by adjacently installed modules. The backplane connects from the LBUSC pins of Slot N to the LBUSA pins of Slot N+1. See Figure B.1. No Local Bus lines are extended beyond Slot 12. Several classes of signal levels are allowed and may be transmitted via this bus. To ensure that modules which produce incompatible levels are not installed in adjacent slots, a keying mechanism has been provided. The Local Bus key provides support for TTL, ECL and analog communication. Each module indicates with the mechanical key on the faceplate which classes of signals it may source or accept nondestructively on each side of its Local Bus. The particular key position for each of the signal classes, as well as sensor-only devices, are specified in the Mechanical Specifications (Section B.7). Sensor-only devices are modules that can non-destructively accept signals of a number of classes, but do not source any signals of their own.

NUMBER	CLASS	-LIMIT	+LIMIT	DRIVE LIMIT
1	TTL	-0.5 V	+5.5 V	200 mA
2	ECL	-5.46 V	0.0 V	50 mA
3	ANALOG LOW	-5.5 V	+5.5 V	50 Ω Drive
4	ANALOG MED	-16.0 V	+16.0 V	500 mA
5	ANALOG HIGH	-42.0 V	+42.0 V	500 mA
6	RESERVED			

TABLE B.3. Logical Classes of LBUS Signals.

RULE B.6.66:
Modules **SHALL NOT** drive more than ±42 V onto any LBUS line.

RULE B.6.67:
Modules **SHALL NOT** drive more than 500 mA DC current into any LBUS line.

RULE B.6.68:
The backplane resistance between any LBUS line and any adjacent pin **SHALL** be greater than 10.0 MΩ.

RULE B.6.69:
The di/dt on any LBUS line **SHALL NOT** exceed ±100 mA/ns.

RECOMMENDATION B.6.6:
Multi-Module instrument systems which utilize the Local Bus for intrasystem communications should be built with anchor and expander protocol. An anchor module has no connections on LBUSA lines. It may drive or receive LBUSC lines. An expander module may drive or receive LBUSA or LBUSC lines. Expanders should be placed in adjacent slots (of higher number) to the anchor. This protocol always guarantees compatibility between adjacent module groups without empty slots.

RULE B.6.70:
Any module that accesses the LBUS lines **SHALL** indicate its class on its local bus keys.

RULE B.6.71:
A module **SHALL NOT** exceed the voltage specified for its class under any loading conditions.

OBSERVATION B.6.12:
If a module provides a direct connection (an electrical short) between LBUSA lines and LBUSC lines, the keying mechanism should indicate that the module will source on LBUSA and LBUSC any class it accepts on either LBUSA or LBUSC respectively.

RULE B.6.72:
A module **SHALL NOT** indicate on its key the RESERVED signal class.

OBSERVATION B.6.13:
The keying mechanism cannot prevent a destructive condition in all situations. It is good practice for a module manufacturer to protect his modules from destruction by any signals within the class it accepts.

RULE B.6.73:
The Slot 0 P2 LBUSA key **SHALL** indicate TTL class.

RULE B.6.74:
The Slot 0 P3 LBUSA and LBUSC keys **SHALL** indicate ECL class.

RULE B.6.75:
The mainframe (if it supports VXIbus Subsystem P2) **SHALL** provide a Slot 0 P2 LBUSC key which indicates TTL class.

RULE B.6.76:
The mainframe (if it supports VXIbus Subsystem P3) **SHALL** provide a Slot 0 P3 LBUSC key which indicates ECL class.

PERMISSION B.6.2:
A device which non-destructively senses signals up to ±16 volts, but does not source any signals or provide direct connection between LBUSA and LBUSC, **MAY** provide a single local bus key tab for ±16 V sensor-only operation.

PERMISSION B.6.3:
A device which non-destructively senses all classes of signals up to ±42 volts, but does not source any signals or provide direct connection between LBUSA and LBUSC, **MAY** provide no local bus key tabs for ±42 V sensor-only operation. Such a device can not cause a destructive condition.

OBSERVATION B.6.14:
A device that does not access the local bus does not provide any local bus keys.

OBSERVATION B.6.15:
Size A, B, and C modules do not provide P3 local bus keys.

B.6.2.7 RSV2-3 (RESERVED)

These pins are reserved.

RULE B.6.77:
Modules and backplanes **SHALL NOT** make any connections to RSV pins.

B.6.3 The VXIbus Subsystem P3 Connector

For higher performance instrumentation, the VXIbus also defines the P3 connector. As with P2, alternate pin definitions of P3 may coexist in VXIbus systems, but any one VXIbus subsystem must exist in contiguous slots. Again, Slot 0 plays a unique role in delivering system resources on P3, such as high speed clocking and trigger routing. In particular, VXIbus P3 adds:

Additional +5V, -5.2V, -2V, ±24V, and ±12V power.

100 MHz differential clock which is synchronous with the P2 10 MHz clock.

A synchronizing signal for 100 MHz clock edge selection.

4 additional ECL trigger lines, for a total of 6.

24 additional local bus lines, for a total of 36.

"Star Trigger" lines for precision module to module timing.

B.6.3.1 CLK100

CLK100 is a 100 MHz system clock. It is sourced from Slot 0 and distributed to slots 1-12 on P3. The Slot 0 CLK100 output is differential ECL. It is buffered on the backplane and distributed to each module slot as a single source, single destination differential ECL signal in a manner similar to CLK10. The distribution delays are matched to provide a tight timing relationship between modules. The backplane buffering provides a high level of inter-module isolation.

RULE B.6.78:
The CLK100 frequency sourced from Slot 0 **SHALL** be 100 MHz. Its accuracy **SHALL** be equal to or better than ±100 ppm (.01%), over its specified operating temperature and time.

OBSERVATION B.6.16:
A Slot 0 module may generate CLK100 internally, or it may derive CLK100 from an external frequency source.

RULE B.6.79:
CLK100 duty cycle **SHALL** be 50% ±20% when measured at the 50% transition level.

RULE B.6.80:
IF CLK100 is switched between different clock sources,
THEN the minimum clock width, high or low, **SHALL NOT** be less than 2.5 ns or more than 1.0 μs during switching. The minimum time between two successive transitions of the same polarity **SHALL NOT** be less than 9.5 ns.

PIN NUMBER	ROWa SIGNAL MNEMONIC	ROWb SIGNAL MNEMONIC	ROWc SIGNAL MNEMONIC	PIN NUMBER
1	ECLTRG2	+24V	+12V	1
2	GND	-24V	-12V	2
3	ECLTRG3	GND	RSV4	3
4	-2V	RSV5	+5V	4
5	ECLTRG4	-5.2V	RSV6	5
6	GND	RSV7	GND	6
7	ECLTRG5	+5V	-5.2V	7
8	-2V	GND	GND	8
9	LBUSA12	+5V	LBUSC12	9
10	LBUSA13	LBUSC15	LBUSC13	10
11	LBUSA14	LBUSA15	LBUSC14	11
12	LBUSA16	GND	LBUSC16	12
13	LBUSA17	LBUSC19	LBUSC17	13
14	LBUSA18	LBUSA19	LBUSC18	14
15	LBUSA20	+5V	LBUSC20	15
16	LBUSA21	LBUSC23	LBUSC21	16
17	LBUSA22	LBUSA23	LBUSC22	17
18	LBUSA24	-2V	LBUSC24	18
19	LBUSA25	LBUSC27	LBUSC25	19
20	LBUSA26	LBUSA27	LBUSC26	20
21	LBUSA28	GND	LBUSC28	21
22	LBUSA29	LBUSC31	LBUSC29	22
23	LBUSA30	LBUSA31	LBUSC30	23
24	LBUSA32	+5V	LBUSC32	24
25	LBUSA33	LBUSC35	LBUSC33	25
26	LBUSA34	LBUSA35	LBUSC34	26
27	GND	GND	GND	27
28	STARX+	-5.2V	STARY+	28
29	STARX-	GND	STARY-	29
30	GND	-5.2V	-5.2V	30
31	CLK100+	-2V	SYNC100+	31
32	CLK100-	GND	SYNC100-	32

TABLE B.4. P3 Pin Definitions.

RULE B.6.81:
CLK100 **SHALL** be synchronous to CLK10.

OBSERVATION B.6.17:
The absolute timing relationship between CLK10 and CLK100 is not specified. This relationship may be produced by deriving CLK10 from CLK100 or deriving CLK100 from CLK10.

PIN NUMBER	ROWa SIGNAL MNEMONIC	ROWb SIGNAL MNEMONIC	ROWc SIGNAL MNEMONIC	PIN NUMBER
1	ECLTRG2	+24V	+12V	1
2	GND	-24V	-12V	2
3	ECLTRG3	GND	RSV4	3
4	-2V	RSV5	+5V	4
5	ECLTRG4	-5.2V	RSV6	5
6	GND	RSV7	GND	6
7	ECLTRG5	+5V	-5.2V	7
8	-2V	GND	GND	8
9	STARY12+	+5V	STARX01+	9
10	STARY12-	STARY01-	STARX01-	10
11	STARX12+	STARX12-	STARY01+	11
12	STARY11+	GND	STARX02+	12
13	STARY11-	STARY02-	STARX02-	13
14	STARX11+	STARX11-	STARY02+	14
15	STARY10+	+5V	STARX03+	15
16	STARY10-	STARY03-	STARX03-	16
17	STARX10+	STARX10-	STARY03+	17
18	STARY09+	-2V	STARX04+	18
19	STARY09-	STARY04-	STARX04-	19
20	STARX09+	STARX09-	STARY04+	20
21	STARY08+	GND	STARX05+	21
22	STARY08-	STARY05-	STARX05-	22
23	STARX08+	STARX08-	STARY05+	23
24	STARY07+	+5V	STARX06+	24
25	STARY07-	STARY06-	STARX06-	25
26	STARX07+	STARX07-	STARY06+	26
27	GND	GND	GND	27
28	STARX+	-5.2V	STARY+	28
29	STARX-	GND	STARY-	29
30	GND	-5.2V	-5.2V	30
31	CLK100+	-2V	SYNC100+	31
32	CLK100-	GND	SYNC100-	32

TABLE B.5. P3 Slot 0 Pin Definitions.

RULE B.6.82:

The CLK100 and SYNC100 backplane distribution network **SHALL NOT** insert more than 2.0 ns timing skew between CLK100 and SYNC100 at any slot.

OBSERVATION B.6.18:

If a 10H101 distribution buffer is used for CLK100 or SYNC100, the trace lengths should be matched within 1 ns (approximately 15 cm (6 inches)).

RULE B.6.83:
Each slot CLK100 **SHALL** be driven by a unique backplane buffer output.

RULE B.6.84:
CLK100 **SHALL** be distributed differentially from Slot 0.

RULE B.6.85:
The backplane CLK100 distribution traces **SHALL** be designed for 50 Ω.

RULE B.6.86:
The CLK100 backplane distribution network **SHALL NOT** insert more than 2.0 ns timing skew between CLK100 signals at any two slots.

RULE B.6.87:
The absolute delay of CLK100 from Slot 0 to any module **SHALL NOT** exceed 8 ns.

RULE B.6.88:
IF a module accesses the CLK100 signals
THEN it **SHALL** provide 50 Ω termination on CLK100+ and CLK100-, with no more than two equivalent ECL loads.

RECOMMENDATION B.6.7:
Use a 10H101 to buffer and fan out CLK100.

B.6.3.2 SYNC100

The synchronization signal SYNC100, is used to synchronize multiple devices with respect to a given rising edge of CLK100. This provides very tight time coordination between modules. The SYNC100 signal is distributed from Slot 0 to slots 1-12, with individual backplane buffers for each slot. The SYNC100 signal can be compared to the GPIB Group Execute Trigger command in function, but with greatly improved time coordination. As the Slot 0 module drives the SYNC100 and CLK100 signals, it may also may be programmed to drive one or more ECLTRG lines in order to qualify the SYNC100 action. An example of this protocol is the ESTST protocol described under ECLTRG Trigger Protocols.

A Slot 0 module implementing the SYNC100 function will provide an arbiter to synchronize external events to CLK100 and meet the guaranteed setup and hold times for the SYNC100 signal. This guarantees that all affected modules will trigger on the same CLK100 clock edge. SYNC100 is a nominally 10 ns pulse and may be initiated by any type of external or internal event: an external BNC trigger, a software register write, the state of any of the TTLTRG*, ECLTRG, LBUS or STAR lines, or any other signal synchronized by the Slot 0 arbiter. Refer to the timing diagram on Figure B.13.

OBSERVATION B.6.19:
SYNC100 delay matching relative to CLK100 is called out under CLK100 matching rules.

RECOMMENDATION B.6.8:
Use a 10H101 to buffer and fan out SYNC100.

RULE B.6.89:
Slot 0 modules **SHALL** provide ≥4.0 ns of setup time from SYNC100 to the rising edge of CLK100.

RULE B.6.90:
Slot 0 modules **SHALL** provide ≥3.5 ns of hold time from rising edge of CLK100 to SYNC100.

RECOMMENDATION B.6.9:
Slot 0 modules which provide SYNC100 support should be designed such that the SYNC100 recycle rate is ≤50 ns.

RULE B.6.91:
Each module slot SYNC100 **SHALL** be driven by a unique backplane buffer output.

RULE B.6.92:
SYNC100 **SHALL** be distributed differentially from Slot 0.

RULE B.6.93:
The backplane SYNC100 distribution traces **SHALL** be designed for 50 Ω.

RULE B.6.94:
The SYNC100 backplane distribution network **SHALL NOT** insert more than 2.0 ns timing skew between SYNC100 signals at any two slots.

RULE B.6.95:
IF a module accesses the SYNC100 signals
THEN it **SHALL** provide 50 Ω termination on SYNC100+ and SYNC100-, with no more than two equivalent ECL loads.

B.6.3.3 STARX and STARY

STARX and STARY provide inter-module asynchronous communication. Two STAR lines are connected between each module slot and Slot 0. Slot 0 may provide a cross matrix switch which can programmably route signals between any two STARX or STARY lines. It may also broadcast a signal received on one STAR line to a group of STAR lines. The STAR lines are bidirectional, providing additional flexibility.

RULE B.6.96:
The STARX and STARY backplane distribution network **SHALL NOT** insert more than 2.0 ns timing skew between any two STAR signals.

RULE B.6.97:
STARX and STARY **SHALL** be driven differentially.

RULE B.6.98:
The backplane STAR distribution traces **SHALL** be designed for 50 Ω.

RULE B.6.99:
IF a module accesses the STAR signals
THEN it **SHALL** provide 50 Ω termination on STAR+ and STAR-, with no more than a single equivalent ECL driver load and a single equivalent ECL receiver load.

RULE B.6.100:
Modules which can both drive and receive STARX or STARY **SHALL** use bidirectional ECL drivers, e.g. 10H123. When the module is programmed to receive, the STAR+ and STAR- drivers **SHALL** be forced into the high impedance state (Vol ≤ -2.0 V)

RULE B.6.101:
A D-size Slot 0 module **SHALL** provide a default differential signal to all STARX and STARY lines which are not being driven by another module.

PERMISSION B.6.4:
If a Slot 0 module does not support STAR bus routing, it may meet the above rule by connecting a 50 Ω resistor between STAR+ lines and ground.

RULE B.6.102:
The absolute delay of any STAR line between Slot 0 and any module **SHALL NOT** exceed 5 ns.

RULE B.6.103:
The backplane **SHALL NOT** connect to the STARX+, STARX-, STARY+, and STARY- pins of the Slot 0 backplane connector.

PERMISSION B.6.5:
The Slot 0 module **MAY** access the STARX+, STARX-, STARY+, and STARY- pins.

OBSERVATION B.6.20:
Allowing all modules, including Slot 0, to access the STAR lines allows construction of modules that may operate in any slot position, including Slot 0.

B.6.3.4 ECLTRG2-5

These lines are functionally the same as the ECLTRG0-1 lines on P2 described in Section B.6.2.4, "ECLTRG0-1", and meet the same rules.

B.6.3.5 LBUS12-35

These lines are functionally the same as the LBUS00-11 lines on P2 described in Section B.6.2.6, "Local Bus", and meet the same rules.

B.6.3.6 RSV4-7

These lines are functionally the same as the RSV2-3 lines on P2 described in Section B.6.2.7, "RSV2-3", and meet the same rules.

B.6.4 Backplane

The backplane is the common signal and power distribution network for all modules. As such, it must be very well designed so as not to introduce noise into the system.

OBSERVATION B.6.21:
Noise should be minimized on the ECLTRG Lines, Clock lines, and SUMBUS lines. It is desirable to maintain time jitter to ≤25 ps. This requires that noise be limited to ±9 mV, assuming an ECL slew rate of 2.75 V/ns.

RULE B.6.104:
All backplane ECL transmission paths **SHALL** be designed for 50 Ω nominal characteristic impedance.

OBSERVATION B.6.22:
When TTL lines share a layer of the backplane with 50 Ω ECL transmission lines, the TTL signal characteristic impedance may also be near 50 Ω. This should provide satisfactory operation (See the VMEbus Specification, Rev. C.1, Observation 6.13).

SUGGESTION B.6.2:
See the MECL System Design Handbook, Motorola, for more backplane and microstrip design information.

B.6.4.1 ECL SIGNAL LEVELS

The signal levels present on the backplane ECL lines are summarized in Table B.6.

Symbol	Parameter	Minimum	Typical	Maximum	Units
V_{OH}	Output logic level "1" (HIGH)	-0.980	-0.9	-0.735	Volts
V_{OL}	Output logic level "0" (LOW)	-1.950	-1.75	-1.630	Volts
V_{BB}	Logic Switching Threshold		-1.29		Volts

TABLE B.6. Backplane ECL signal levels.

B.7 MECHANICAL SPECIFICATIONS

B.7.1 Introduction

Information is provided in this chapter to ensure that VXIbus modules, backplanes, mainframes, and associated mechanical accessories are dimensionally compatible.

The mechanical dimensions given in this chapter are based on ANSI/IEEE 1101 and IEC publications 297-3 and 603-2. The electrical characteristics for VXIbus connectors, as specified here, supersede publication 603-2 where they differ.

IEC publication 603-2 describes a family of connector types which are identified by labels of the form:

603-2-IEC-xxxxxx-xxx

All of the P1/J1, P2/J2 and P3/J3 connectors used on VXIbus modules and backplanes are members of this family. In this chapter, the label 603-2-IEC-xxxxxxx-xxx is used when referring to all of these connector types as a group. The label 603-2-IEC-C096Mx-xxx is used when referring to the 96-pin male connector types within this family which are used on the VXIbus modules. 603-2-IEC-C096Fx-xxx is used when referring to the 96-pin female connector types which are used on VXIbus backplanes.

Figure B.14 is a front view of a typical VXIbus mainframe. In the mainframe shown, modules are inserted into the mainframe from the front, in a vertical plane, with the component face of the module board on the right.

B.7.2 Module Specifications

B.7.2.1 VXIbus BOARDS and MODULES

RULE B.7.1:

VXIbus modules **SHALL** fit into one of four size classifications:

1. Single height module *typically* containing a board with dimensions 100 mm (3.937 in) high x 160 mm (6.299 in) deep will be referred to as a "Size A" module.
2. Double height module *typically* containing a board with dimensions 233.35 mm (9.187 in) high x 160 mm (6.299 in) deep will be referred to as a "Size B" module.
3. Double height module *typically* containing a board with dimensions 233.35 mm (9.187 in) high x 340 mm (13.386 in) deep will be referred to as a "Size C" module.
4. Triple height module *typically* containing a board with dimensions 366.7 mm (14.437 in) high x 340 mm (13.386 in) deep will be referred to as a "Size D" module.

RULE B.7.2:

Size A and size B boards and board assemblies **SHALL** be designed according to the dimensions given in the VMEbus specification. [1]

OBSERVATION B.7.1:

Size A and size B boards and board assemblies are designed for a mainframe with 20.32 mm (0.8 in) spacing between slots.

RULE B.7.3:
Size C and size D modules **SHALL** be contained within the envelope specified in Figures B.17, B.18, and B.21.

RULE B.7.4:
Size C and size D modules **SHALL** be designed for a mainframe with 30.48 mm (1.2 in) spacing between slots.

OBSERVATION B.7.2:
Typical size C and size D modules will contain a board as shown in Figures B.15 and B.16.

RULE B.7.5:
Modules **SHALL** provide the board edge feature as specified in Figure B.21 to allow proper insertion into a mainframe. This feature **SHALL** be 1.6 ± 0.2 mm (0.063 ±0.008 in) thick.

RECOMMENDATION B.7.1:
Design C and D modules around a board as specified in Figures B.15 and B.16. This will ensure proper position of connectors, front panel, card edge, etc.

B.7.2.1.1 Board Connectors

Size A modules have only one backplane connector, called the P1 connector. Size B and C modules have either one or two backplane connectors, called the P1 and P2 connectors. Size D modules have one, two, or three connectors, called the P1, P2, and P3 connectors. In all modules, the P1 connector is mandatory, P2 and P3 are optional.

RULE B.7.6:
P1, P2, and P3 connectors **SHALL** meet or exceed the mechanical specifications of an 603-2-IEC-C096Mx-xxx class 2 connector.

OBSERVATION B.7.3:
603-2-IEC class 2 connectors have a minimum mechanical endurance of 400 insertion/extraction cycles.

RULE B.7.7:
The mounting location of the P1, P2, and P3 connectors **SHALL** be consistent with the hole patterns specified in Figures B.15 and B.16.

RULE B.7.8:
The P1, P2, and P3 connectors **SHALL** be positioned as shown in Figures B.19 and B.20.

B.7.2.1.2 Board Assemblies

The board assembly typically consists of a PC board and connectors, as defined above, and electronic components.

RULE B.7.9:
Solder fillets, traces, shielding, and components on VXIbus boards **SHALL NOT** be closer than 2.5 mm (0.098 inch) from the top and bottom edges of the board to guarantee clearance between them and the board guides. Figures B.15 and B.16 show these dimensions for size C and D modules.

RULE B.7.10:
IF a module contains components sensitive to orientation (e.g. mercury switches),
THEN that module's specifications and labels **SHALL** clearly state its limitations.

RECOMMENDATION B.7.2:
An orientation sensitive module should be designed to be installed vertically with its component side to the right, or horizontally with its component side up.

B.7.2.1.3 Module Width

PERMISSION B.7.1:
Size C or D modules which require more than the standard spacing **MAY** be designed to occupy more than one slot, adding an integer multiple of 30.48 mm (1.2 in) to the module width.

OBSERVATION B.7.4:
Component height, lead length, shield positions, etc of the typical module are shown in Figure B.18.

PERMISSION B.7.2:
Module shields as shown in Figure B.17 and Figure B.18 **MAY** be omitted or their positions changed, but must remain within the defined envelope.

RECOMMENDATION B.7.3:
Include module shields to improve electromagnetic compatibility (EMC), ease of handling, etc.

B.7.2.1.4 Board Warpage, Lead Length, Shield Height, and Component Height

RULE B.7.11:
All A and B size board assemblies **SHALL** meet the VMEbus specification 7.2.6.[1]

RULE B.7.12:
For shielded C and D size modules, the outer surface of the solder-side shield, including warpage, **SHALL** be greater than or equal to 1.57 mm (0.062 inch) from the intermodule separation plane when installed. The outer surface of the component-side shield, including warpage, **SHALL** be greater than or equal to 0.15mm (0.006 inch) from the intermodule separation plane when installed. Refer to Figure B.18.

RECOMMENDATION B.7.4:
To account for warpage, the nominal solder-side shield should be greater than 2.07mm (0.082 inch) from the intermodule separation plane. The nominal component-side shield should be greater than 0.65mm (0.026 inch) from the intermodule separation plane. Refer to Figure B.18.

RULE B.7.13:
For unshielded C and D size modules, leads and solder-side components, including warpage, **SHALL** be greater than or equal to 3.96mm (0.156 inch) from the intermodule separation plane when installed. Components, including warpage, **SHALL** be greater than or equal to 2.54mm (0.100 inch) from the intermodule separation plane when installed. Refer to Figure B.18.

B.7.2.2 FRONT PANEL

This section provides the mechanical specifications for the double and triple height module front panels and associated hardware. Figure B.22 shows a double height, single width (C size) front panel. Figure B.23 shows a triple height, single width (D size) front panel.

RULE B.7.14:
All modules **SHALL** include a front panel.

RULE B.7.15:
Each module, regardless of size, **SHALL** have a front panel conforming to the size of the mainframe it is used in.

PERMISSION B.7.3:
Additional hardware **MAY** be utilized to fill a front panel gap when inserting a small module into a larger mainframe.

RULE B.7.16:
Front panel screws **SHALL** be provided to secure the top and bottom of the panel to the mainframe, and their screw threads **SHALL** be M2.5 x 0.45 pitch. See Figure B.22.

RULE B.7.17:
Front panels **SHALL** make contact with chassis ground in the area surrounding the retention screws as shown in Figure B.22 and Figure B.23 This surface **SHALL** be conductive, smooth, and free of all protrusions.

B.7.2.2.1 Front Panel Mounting

RULE B.7.18:
All modules **SHALL** comply with the test dimension shown in Figure B.26 and Figure B.27. This dimension is from the rear face of the front panel to the rear face of the backplane connector.

B.7.2.2.2 Front Panel Dimensions

RULE B.7.19:
Single width C and D size front panels **SHALL** be designed to the dimensions shown in Figure B.22 and Figure B.23 respectively.

RULE B.7.20:
IF a C or D size module occupies more than one slot
THEN the width of the front panel **SHALL** be 30.18 mm (1.188 inch) plus an integral multiple (N) of 30.48 mm (1.200 inch), where N is the number of slots that the module occupies minus one.

RULE B.7.21:
IF local bus lockout keys are required on a module,
THEN the front panel **SHALL** be notched as shown in Figure B.22 and Figure B.23.

SUGGESTION B.7.1:
Include the lockout key notch on all front panels. This will simplify removal of a module when the adjacent module is keyed.

B.7.2.2.3 Filler Panels

RULE B.7.22:
Filler panels **SHALL** be used in any empty slots.

OBSERVATION B.7.5:
Filler panels are often required by safety regulations, and may be necessary for cooling and EMC performance.

RULE B.7.23:
C and D size filler panels **SHALL** conform to dimensions given in Figure B.24 and Figure B.25 respectively.

RULE B.7.24:
A and B size filler panels **SHALL** conform to VMEbus specification 7.3.4.[1]

B.7.2.2.4 Board Injectors/Ejectors

Insertion and extraction forces of the module relative to the mainframe may be very high. These forces are greatest for D size modules that use all three backplane connectors.

RECOMMENDATION B.7.5:
Modules using three or more of the 96 pin 603-2-IEC-C096Mx-xxx connectors should provide an injector/ejector system.

OBSERVATION B.7.6:
The insertion force for three 96 pin 603-2-IEC-C096Mx-xxx connectors can be as high as 270 N (60.69 lbf).

PERMISSION B.7.4:
An injector and /or ejector system **MAY** be provided on any module size.

RULE B.7.25:
Modules equipped with ejectors or injectors **SHALL NOT** interfere with mainframes designed to the VXIbus or VMEbus specification. See Figure B.37 and Figure B.38.

OBSERVATION B.7.7:
Refer to Section B.7.3.3, "Injection/Ejection", for more information

B.7.2.2.5 Front Panel Interface

RULE B.7.26:
External wiring to a module, if any, **SHALL** exit the module only through the front panel area.

OBSERVATION B.7.8:
External wiring is defined as any connectors, cables, etc. that carry signals or interfaces out of the mainframe in which the module is installed. The above rule does not exclude module-to-module wiring.

RULE B.7.27:
Connector and front panel interfaces **SHALL NOT** leave significant gaps in the front panel, or disrupt proper airflow within the system.

OBSERVATION B.7.9:
Although the terms "significant gaps" and "proper air flow" can be rather broadly interpreted, it is intended that a manufacturer be very cautious about the front panel components chosen.

B.7.2.2.6 Front Panel Handles

RULE B.7.28:
Handles **SHALL NOT** interfere with the local bus lockout keys of adjacent modules.

SUGGESTION B.7.2:
Handle width and position should comply with the dimensions shown in Figure B.22 and Figure B.23. This will avoid interference with local bus lockout keys.

OBSERVATION B.7.10:
Injection and ejection constraints may require the use of a handle with dimensions different than those in Figure B.22 and Figure B.23.

B.7.2.3 MODULE SHIELDING

Shields are often included in modules to improve EMC performance, ease of handling, etc. Refer to Figure B.17 and Figure B.18.

PERMISSION B.7.5:
Module shields **MAY** be tied to chassis ground or circuit ground.

RULE B.7.29:
C size and D size modules **SHALL** provide room for the optional mainframe chassis shield as shown in Figure B.17 and Figure B.18.

OBSERVATION B.7.11:
The envelope as defined in Figure B.17 ensures that module shields do not contact adjacent modules.

OBSERVATION B.7.12:
The use of box type plug-in units as specified by IEC 297-3 and ANSI/IEEE 1101 may be limited. Maximum module length, injector surfaces, and non-removable card guides may be incompatible. Refer to Figure B.17, Figure B.21, and Figure B.38.

PERMISSION B.7.6:
A grounding shield **MAY** be provided around each 96 pin connector as shown in Figure B.34. This will be referred to as the connector shield.

OBSERVATION B.7.13:
Connector shield contacts may or may not be removable. However, removable contacts may be advisable for some modules. The module manufacturer should evaluate the applicability of removable contacts for each module.

PERMISSION B.7.7:
A module **MAY** make electrical contact with a compatible adjacent module at any point in the area near the front panel as specified in Figure B.17. This contact may be implemented by means of gasketing, clips, etc.

RULE B.7.30:
IF gasketing is installed for contact with a compatible adjacent module,
THEN it **SHALL** be installed on the solder side only, and it **SHALL** be compressible to fit within the intermodule separation plane.

RECOMMENDATION B.7.6:
Gasketing installed for contact with a compatible adjacent module should be removable. This will allow installation adjacent to unshielded or otherwise incompatible modules.

RECOMMENDATION B.7.7:
The areas specified in Figure B.17 for contact with adjacent modules should be chassis ground and conductive.

B.7.2.4 MODULE COOLING

RULE B.7.31:

Cooling air **SHALL** flow as shown in Figure B.29, Figure B.30, and Figure B.39. Airflow direction is from P3 to P1.

RULE B.7.32:

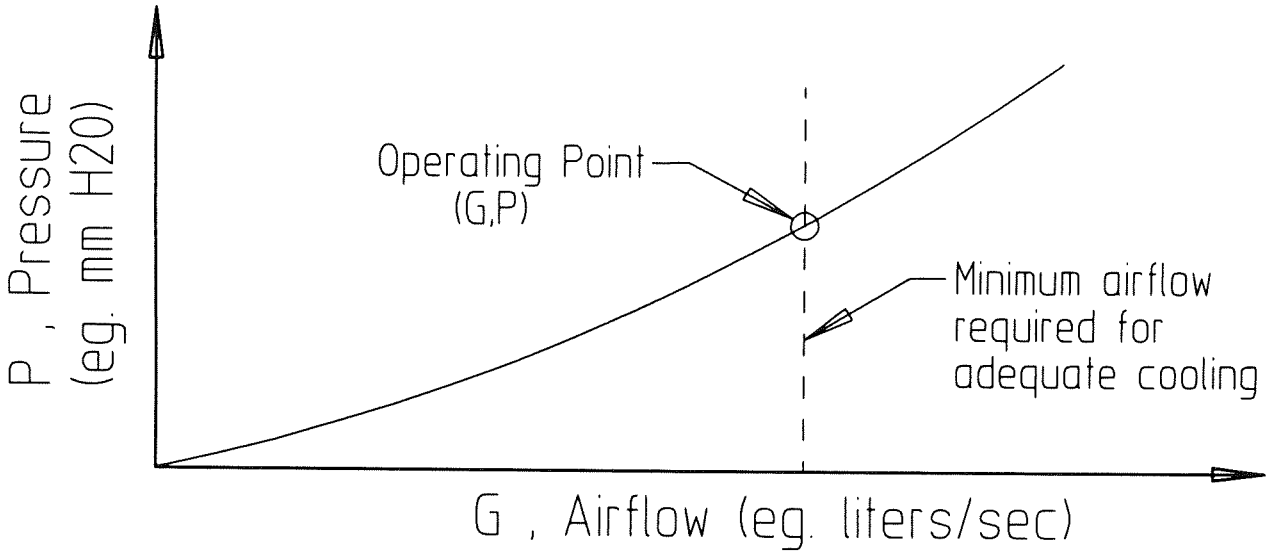
Cooling requirements **SHALL** be established for all modules and included in product specifications. The specification **SHALL** include the maximum ambient operating temperature and the operating point. (Example: 55 °C ambient maximum operating temperature. For 10°C internal rise, 0.5 mm H_2O @ 1.5 liters/sec)

RECOMMENDATION B.7.8:

Label the module with the cooling specification. This will facilitate system integration, see Figure B.42.

RECOMMENDATION B.7.9:

Establish a "resistance" pressure drop vs airflow graph and include in the product specifications. This is useful for more detailed system integration. Example:



OBSERVATION B.7.14:

An example of simplified test set-up and procedures for establishing module cooling requirements is shown in Figure B.40.

OBSERVATION B.7.15:

Great care must be taken with the design of modules that require cooling air to flow in parallel paths. Examples are modules with multiple boards and modules with components on the solder side of the main board. In particular, the power dissipation of solder side components may be limited by the restricted access of cooling air. The module designer is responsible for assuring that resistance to air flow is consistent with the cooling requirements for each parallel path.

B.7.2.5 MODULE POWER

RECOMMENDATION B.7.10:
Attach an abbreviated version of the power supply voltage and current requirements to the module. This will facilitate system integration.

B.7.2.6 MODULE KEYING

Lockout keys are provided to serve as an impediment to local bus incompatibilities.

RULE B.7.33:
IF modules access a local bus,
THEN lockout keys **SHALL** be provided, consistent with the particular bus accessed. The lockout keys **SHALL** conform to the specifications in Figure B.28.

RULE B.7.34:
IF lockout keys are required on a module,
THEN they **SHALL** be installed by the manufacturer, consistent with the definition of the product.

OBSERVATION B.7.16:
Lockout keys are not intended to provide failsafe protection. Accidental damage may still occur if incompatible modules are forced next to each other on the local bus.

B.7.2.7 MODULE ENVIRONMENTAL

RECOMMENDATION B.7.11:
Test modules for temperature, humidity, vibration, shock, etc., according to the procedures described in IEC Publication 68. Include the results in product specifications, along with module mass.

OBSERVATION B.7.17:
It is a system integrator's responsibility to select modules and mainframes appropriate for the application's environmental requirements.

B.7.3 Mainframe Specifications

RULE B.7.35:
VXIbus mainframes **SHALL** fit into one of four size classifications:

1. A mainframe that provides the J1 backplane and allows the insertion of an A-size module, only, will be referred to as a "A-size" mainframe, and conform to the VMEbus specifications[1] for subracks.
2. A mainframe that allows the insertion of a B-size module, maximum, will be referred to as a "B-size" mainframe, and conform to the VMEbus specifications[1] for subracks. The J1 backplane is mandatory, J2 is optional.
3. A mainframe that allows the insertion of a C-size module, maximum, will be referred to as a "C-size" mainframe. The J1 backplane is mandatory, J2 is optional.
4. A mainframe that allows the insertion of a D-size module, maximum, will be referred to as a "D-size" mainframe. The J1 backplane is mandatory, J2 and J3 are optional.

RECOMMENDATION B.7.12:
Allowances should be made for the insertion of A size modules into B size mainframes, the insertion of A and B size modules into C size mainframes, and the insertion of A, B, and C size modules into D size mainframes.

OBSERVATION B.7.18:
Additional hardware and removable card guides may be required to allow the insertion of smaller modules into a larger mainframe.

RULE B.7.36:
IF the insertion of smaller modules is allowed into a larger mainframe,
THEN the mainframe manufacturer **SHALL** specify a procedure and/or means for the insertion of the smaller modules.

PERMISSION B.7.8:
Mechanical and/or electrical means may be used for the adaptation of a smaller module to a larger mainframe.

RULE B.7.37:
C and D size mainframes **SHALL** be designed for 30.48 mm (1.2 in) slot spacing, as shown in Figure B.29 and Figure B.30.

RULE B.7.38:
A and B size mainframes **SHALL** be designed to 20.32 mm (0.8 inch) spacing.

RULE B.7.39:
C and D size mainframes **SHALL** conform to the dimensions and tolerances shown in Figure B.29 and Figure B.30. This ensures that connectors mate properly and that modules will fit in the mainframe without interference.

RULE B.7.40:
With the mainframe oriented with the modules vertical, P1 at the top, component side of module boards to the right; Slot 0 **SHALL** be the left most slot of any VXIbus subsystem.

OBSERVATION B.7.19:
This rule does not mandate mainframe orientation, it only establishes the position of Slot 0 relative to mainframe orientation.

SUGGESTION B.7.3:
IF vertical orientation of the modules is desired,
THEN Slot 0 should be at the left.
IF horizontal orientation of the modules is desired,
THEN Slot 0 should be at the bottom.

RULE B.7.41:
Card guides **SHALL** be made of non-conductive material, or be isolated from chassis ground.

OBSERVATION B.7.20:
Removable card guides may permit the use of box style plug-in units as specified by IEC 297-3 and ANSI/IEEE 1101.

B.7.3.1 BACKPLANES

RULE B.7.42:
Each backplane **SHALL** be a single monolithic board, within any slot position.

RULE B.7.43:
Backplane connectors for C and D size mainframes **SHALL** be positioned as shown in Figures B.31 and B.32.

PERMISSION B.7.9:
Backplanes **MAY** deviate from Figure B.31 and Figure B.32 in structure and mounting methods.

OBSERVATION B.7.21:
Typical C and D size backplanes are shown in Figure B.31 and Figure B.32 for reference. Adherence to these drawing will ease the task of mounting and aligning the backplane within the mainframe.

RULE B.7.44:
IF components other than connectors are placed on the component side of a size C or D backplane,
THEN they **SHALL** be placed outside the shaded areas specified in Figure B.33. Non-conductive components **SHALL** be less than 12.2 mm (0.480 inch) in height. Conductive components **SHALL** be less than 10.0 mm (0.394 inch) in height.

RECOMMENDATION B.7.13:
Mainframes should allow shielding around each 96-pin backplane connector. This shield typically provides a connection between a ground ring on the backplane and a module. See the references to the "connector shield" in Section B.7.2.3, "Module Shielding". Refer also to the above Rule, Figure B.33, and Figure B.34.

RULE B.7.45:
Connector shields **SHALL** comply with Figure B.34.

RULE B.7.46:
IF conductive backplane traces are exposed within the ground ring area defined by Figure B.33,
THEN the potential of those traces **SHALL** be circuit ground.

RULE B.7.47:
Components **SHALL NOT** be located near backplane connectors as specified in Figure B.33.

RULE B.7.48:
The 96 pin 603-2-IEC-C096Fx-xxx connectors **SHALL** be used on all J1, J2, and J3 positions.

RULE B.7.49:
The VXIbus J1 bus **SHALL** have some provision for jumpering the interrupt acknowledge and bus grant daisy-chains when boards are not plugged into a slot.

B.7.3.2 GROUNDING

OBSERVATION B.7.22:
Mainframes may connect circuit ground to chassis ground.

RULE B.7.50:
All mainframes **SHALL** provide chassis ground to the module front panels in the area of the front panel mounting screws. The chassis ground contact area is shown in Figure B.22 for C size modules, and in Figure B.23. for D size modules.

B.7.3.3 INJECTION/EJECTION

RECOMMENDATION B.7.14:
Mainframes should provide a surface for module extraction as shown in Figure B.37.

RECOMMENDATION B.7.15:
Mainframes should provide a surface for module injection, as shown in Figure B.38.

RULE B.7.51:
Mainframes **SHALL NOT** interfere with injector/ejector systems conforming to Figures B.37 and B.38.

B.7.3.4 MAINFRAME SHIELDING

PERMISSION B.7.10:
Mainframes **MAY** allow the installation of a chassis potential shield as shown in Figure B.17, and Figure B.18.

RULE B.7.52:
IF mainframe chassis shields are provided,
THEN they **SHALL** be removable.

RULE B.7.53:
IF the shield potential of a module is not chassis ground, or the module is unshielded,
THEN any adjacent chassis shield **SHALL** be insulated.

PERMISSION B.7.11:
IF the shield potential of a module is defined as chassis ground,
THEN electrical contact between the module and adjacent chassis shields **MAY** be allowed.

RECOMMENDATION B.7.16:
IF a mainframe has provision for chassis shields,
THEN both insulated and non insulated shields should be available.

B.7.3.5 MAINFRAME COOLING

RULE B.7.54:
IF a mainframe provides cooling air,
THEN it **SHALL** supply the air to each slot in the minimum area specified in Figure B.39.

RULE B.7.55:
IF a mainframe provides cooling air,
THEN that mainframe **SHALL** provide room for the exhausted cooling air to exit each slot in the minimum area specified in Figure B.39.

PERMISSION B.7.12:
A mainframe **MAY** provide for additional cooling air entry and exhaust in areas outside those specified in Figure B.39.

RULE B.7.56:
Cooling specifications **SHALL** be established for all mainframes and included in the product specifications. These specifications **SHALL** include a plot of "worst slot" pressure drop vs flow rate through the slot. See Figure B.42.

RECOMMENDATION B.7.17:
Label the mainframe with a copy of the "worst slot" graph. This will facilitate system integration. See Figure B.42.

OBSERVATION B.7.23:
"Worst slot" refers to worst case airflow through a slot vs the pressure drop imposed by a module in the slot. In most cases, this means identifying the slot with the least airflow, and testing it with all other slots empty.

OBSERVATION B.7.24:
Airflow can often be increased by blocking unused slots. This technique is not valid when testing the "worst slot", however, as the presence of unused slots cannot be assumed.

SUGGESTION B.7.4:
Establish plots of pressure drop vs airflow for each slot in the mainframe and include them in the product specifications.

OBSERVATION B.7.25:
Airflow plots of each slot and the entire mainframe are useful for more detailed system integration.

OBSERVATION B.7.26:
The cooling specifications required by VXI are sufficient for most system integration. Some systems may require more detailed analysis and specification. Examples are variable or high power systems, proprietary systems, borderline cases, etc.

OBSERVATION B.7.27:
An example of simplified test set-up and procedures for establishing mainframe cooling specifications is shown in Figure B.41.

B.7.3.6 MAINFRAME POWER

RECOMMENDATION B.7.18:
Label the mainframe with an abbreviated version of the power supply specifications. This label should specify available current of the supply voltages, and state whether circuit ground is connected to chassis ground. This will facilitate system integration.

B.7.3.7 KEYING

RULE B.7.57:
Mainframes **SHALL** provide local bus lockout keys for Slot 0. These keys **SHALL** allow only TTL compatible modules on P2 and only ECL compatible modules on P3. See Figure B.28.

OBSERVATION B.7.28:
Mainframe lockout keys typically consist of a tab or recess to the left of Slot 0. See Figure B.14.

B.7.3.8 MAINFRAME ENVIRONMENTAL

RECOMMENDATION B.7.19:
Test mainframes for temperature, humidity, vibration, shock, etc., according to the procedures described in IEC Publication 68. Include the results in product specifications.

SUGGESTION B.7.5:

Include specifications for the vibration and shock environment experienced by modules installed in a mainframe during testing of the mainframe. This should include a variety of module weights and slot positions.

OBSERVATION B.7.29:

It is a system integrator's responsibility to select modules and mainframes appropriate for the application's environmental requirements.

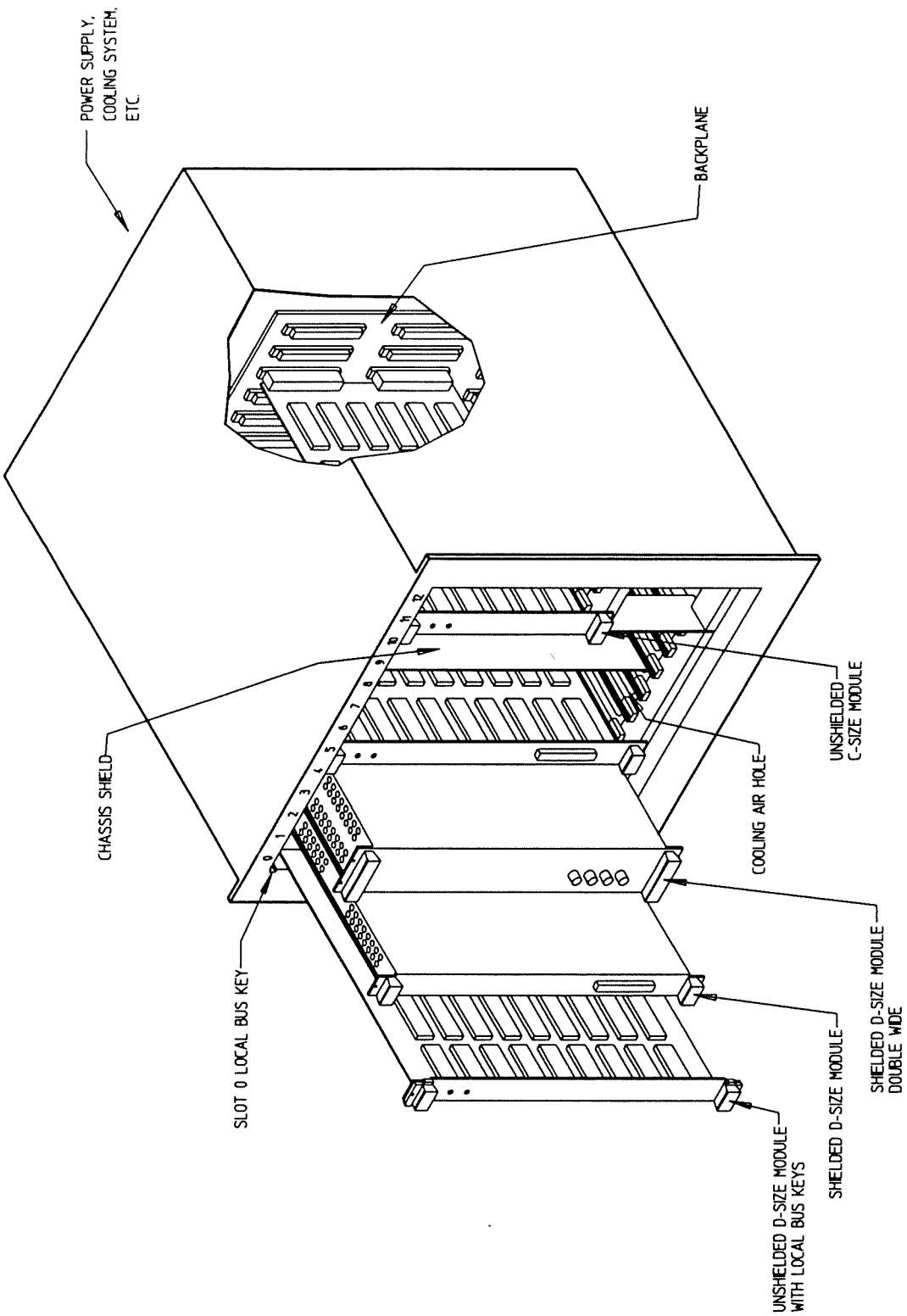
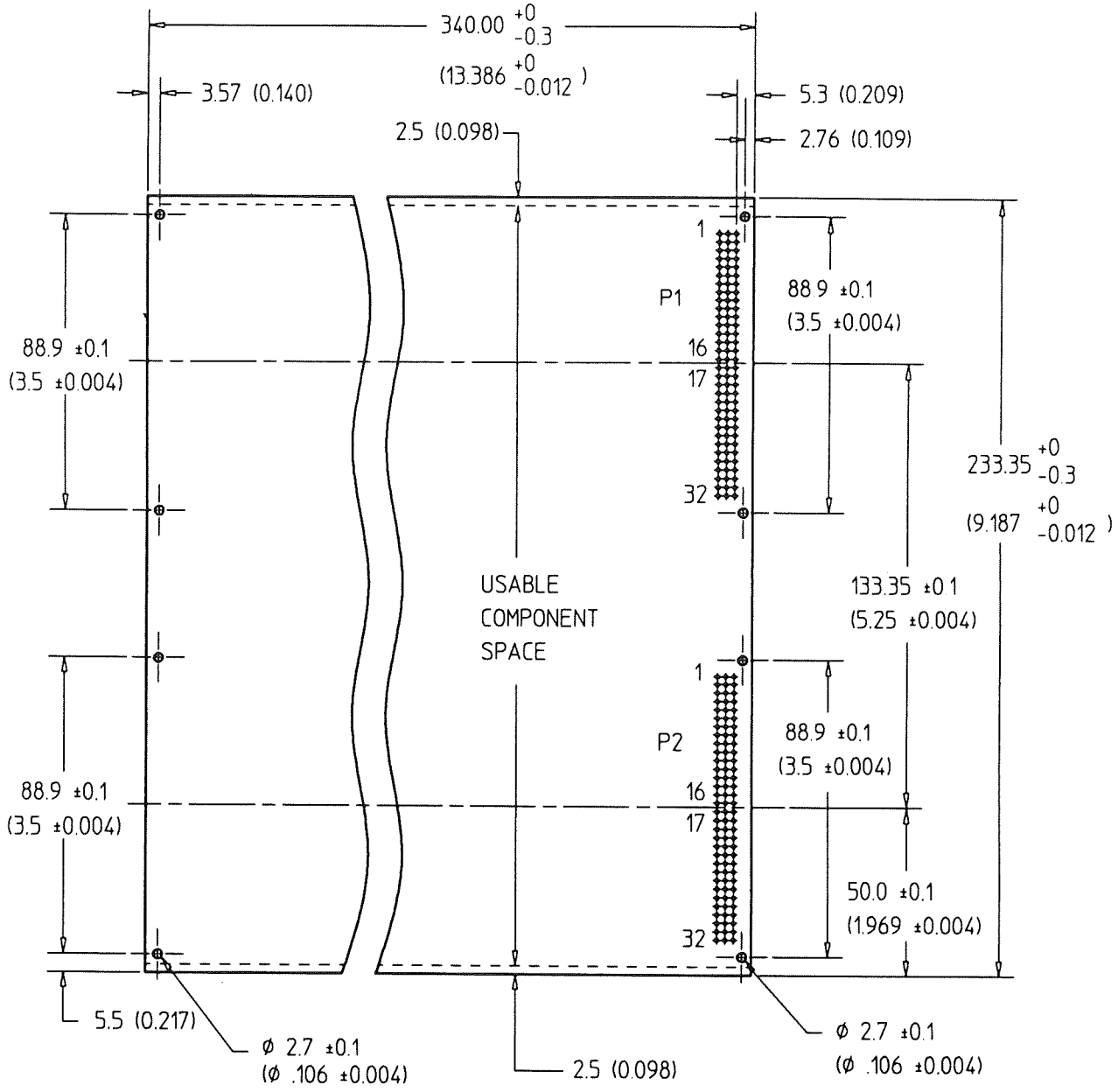


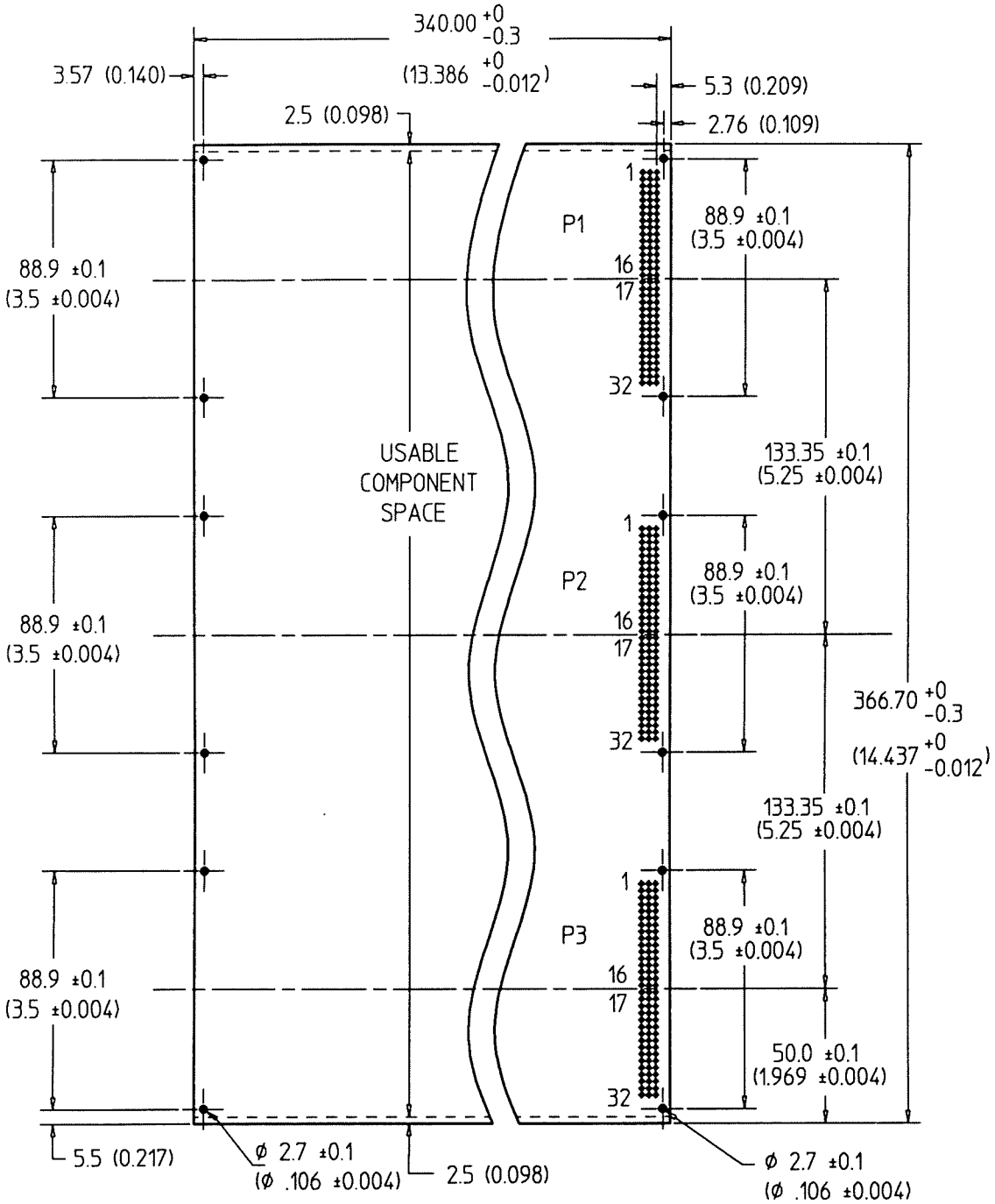
Figure B.14. Typical D-size mainframe.



NOTES:

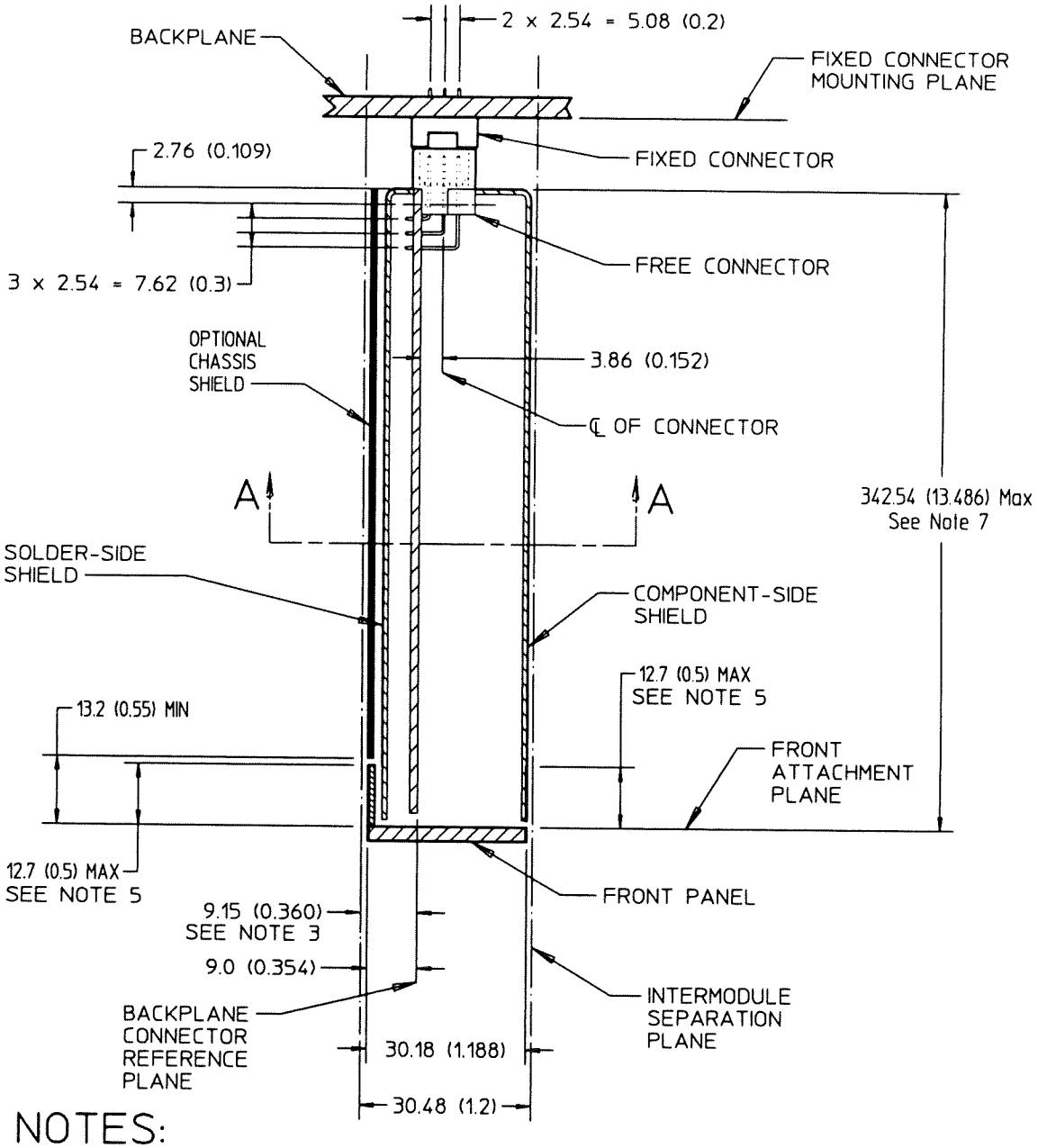
1. All dimensions are shown in millimeters.
   Inch dimensions are shown in parentheses.
2. Front panel mounting holes are suggestions only.
3. OBSERVATION: P1 is required for VXI devices. P2 is optional.
4. OBSERVATION: Board outline may be reduced to accommodate shields.

Figure B.15. Size C board



NOTES:
1. All dimensions are shown in millimeters.
   Inch dimensions are shown in parentheses.
2. Front panel mounting holes are suggestions only.
3. OBSERVATION: P1 is required for VXI devices. P2 and P3 are optional.
4. OBSERVATION: Board outline may be reduced to accommodate shields.

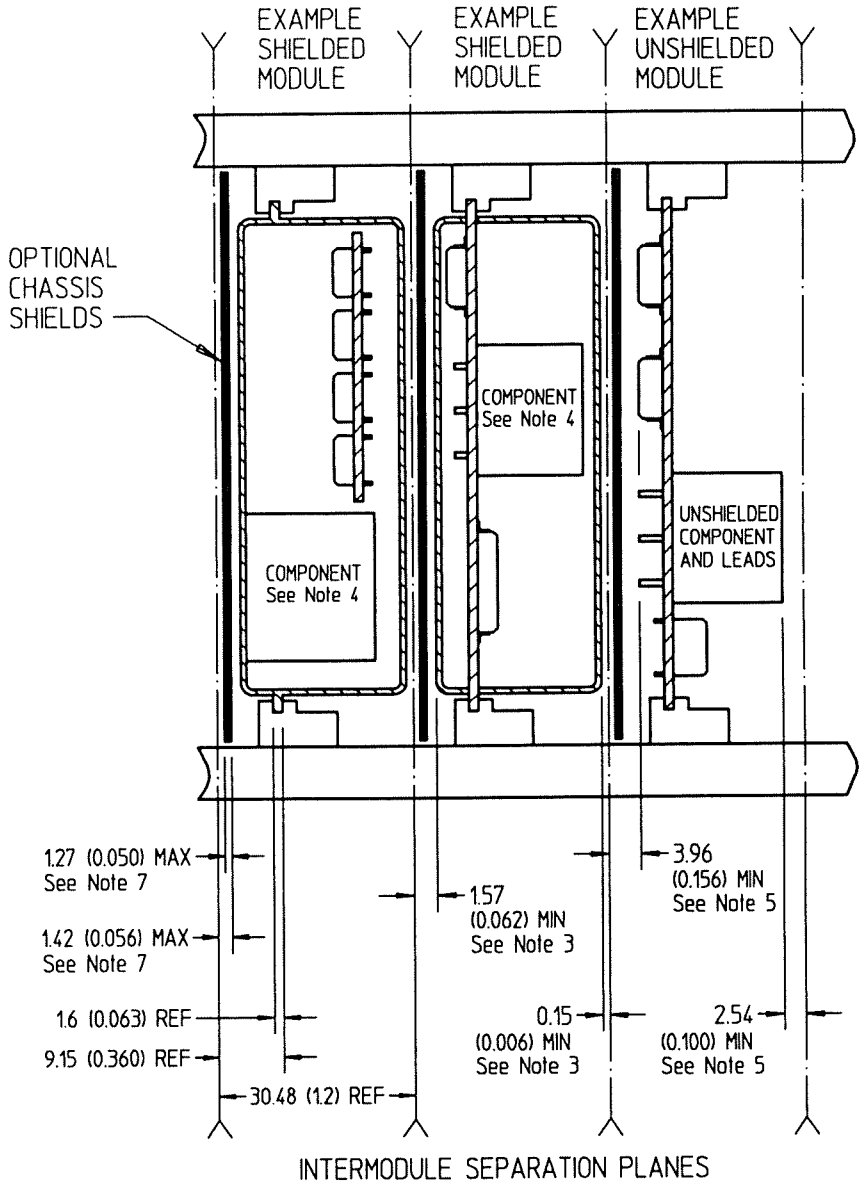
Figure B.16. Size D board



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. For additional information regarding the allowed thickness of boards, refer to the section "Module Width".
3. This 9.15 mm (0.360 inch) dimension is the same regardless of the thickness of the board.
4. Refer to the section "Component Height, Lead Length, Shield Height and Warpage".
5. This area may be used for electrical contact with compatible adjacent modules. Refer to the section "Module Shielding".
6. View A-A. See figure "Module Envelope, Front View" and figure "Module Edge Guide Feature".
7. Refer to the section "Backplanes" rule on maximum component height, and the section "Module Shielding" observation on box type plug-in units.

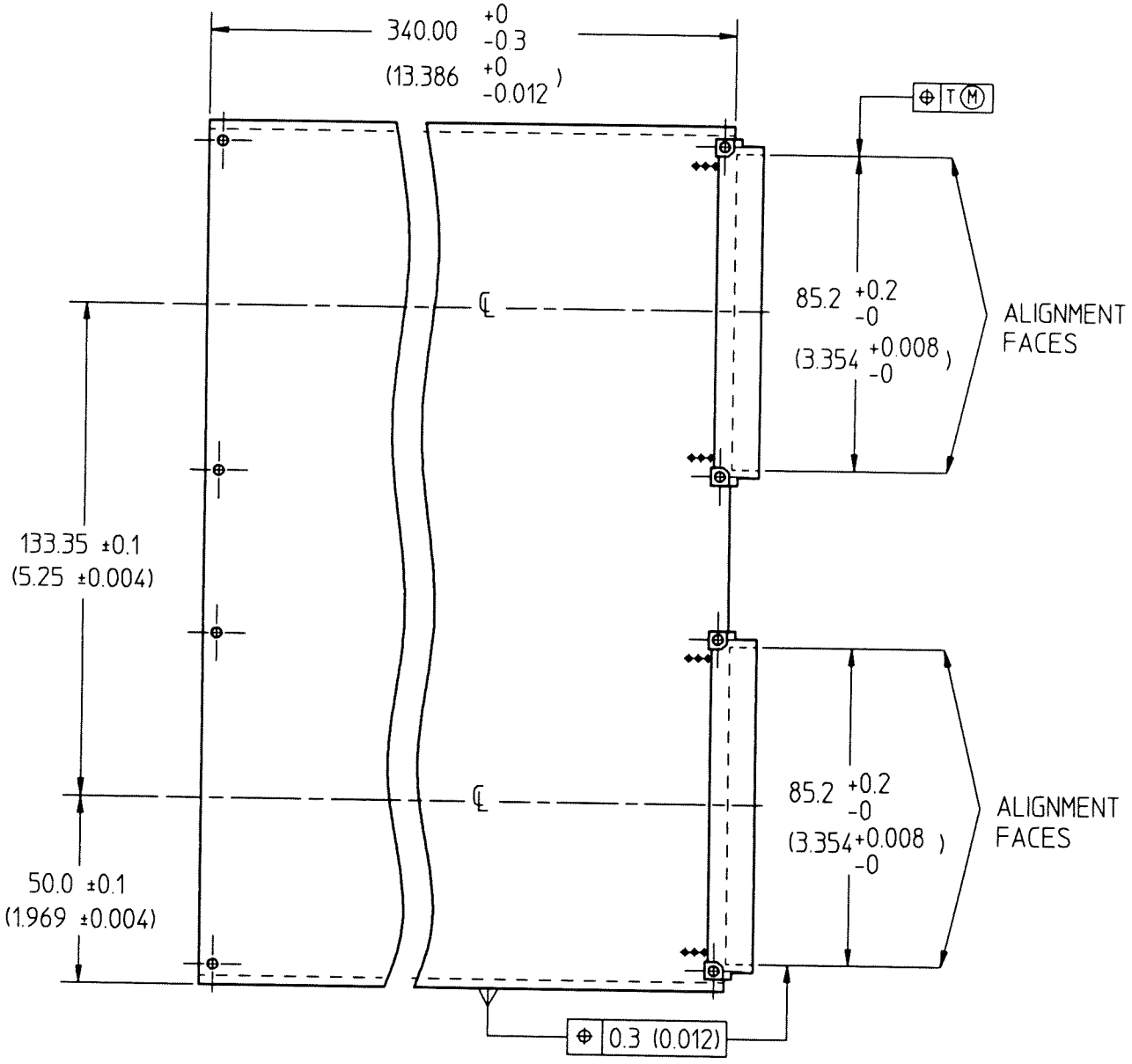
Figure B.17. Module envelope, top view


EXAMPLE SHIELDED MODULE
EXAMPLE SHIELDED MODULE
EXAMPLE UNSHIELDED MODULE
OPTIONAL CHASSIS SHIELDS
COMPONENT See Note 4
COMPONENT See Note 4
UNSHIELDED COMPONENT AND LEADS
1.27 (0.050) MAX See Note 7
1.42 (0.056) MAX See Note 7
1.6 (0.063) REF
9.15 (0.360) REF
30.48 (1.2) REF
1.57 (0.062) MIN See Note 3
0.15 (0.006) MIN See Note 3
3.96 (0.156) MIN See Note 5
2.54 (0.100) MIN See Note 5
INTERMODULE SEPARATION PLANES

NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. Refer to the section, "Component Height, Lead Length, Shield Height and Warpage".
3. Minimum shield clearance refers to module when installed and includes the effects of warpage. RECOMMENDATION: Nominal clearance should be at least 0.5 (0.020) greater.
4. No restrictions are placed on components inside a shielded module.
5. Applies only to unshielded components and leads.
6. OBSERVATION: Extreme care must be taken with unshielded modules. Safety, handling, voltage, warpage, etc. should be considered.
7. RULE: IF provided, THEN the chassis shield MUST conform to these dimensions.

Figure B.18. Module envelope, front view



NOTES:

1. All dimensions are shown in millimeters.
   Inch dimensions are shown in parentheses.

Figure B.19. C-size board assembly connector positions.

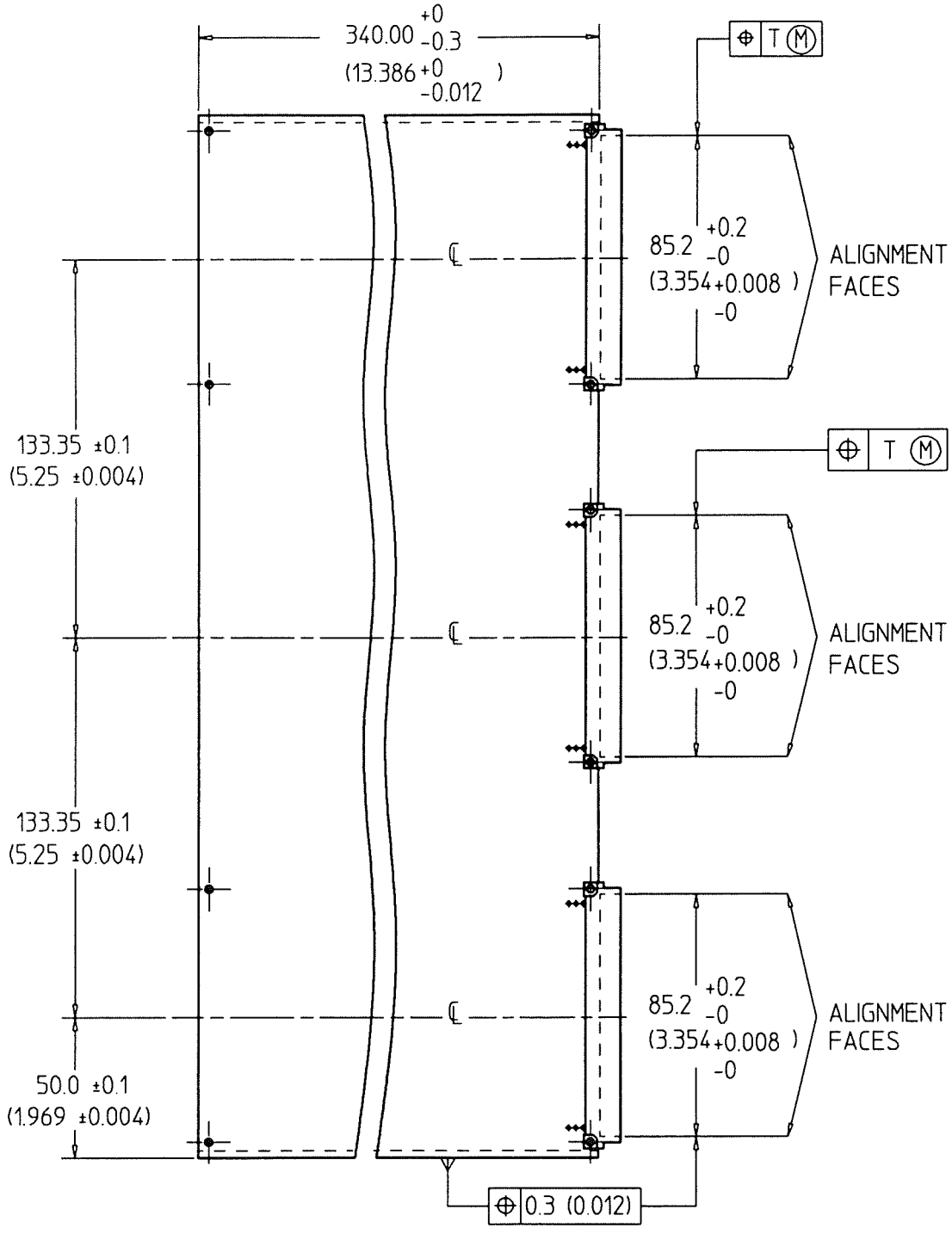
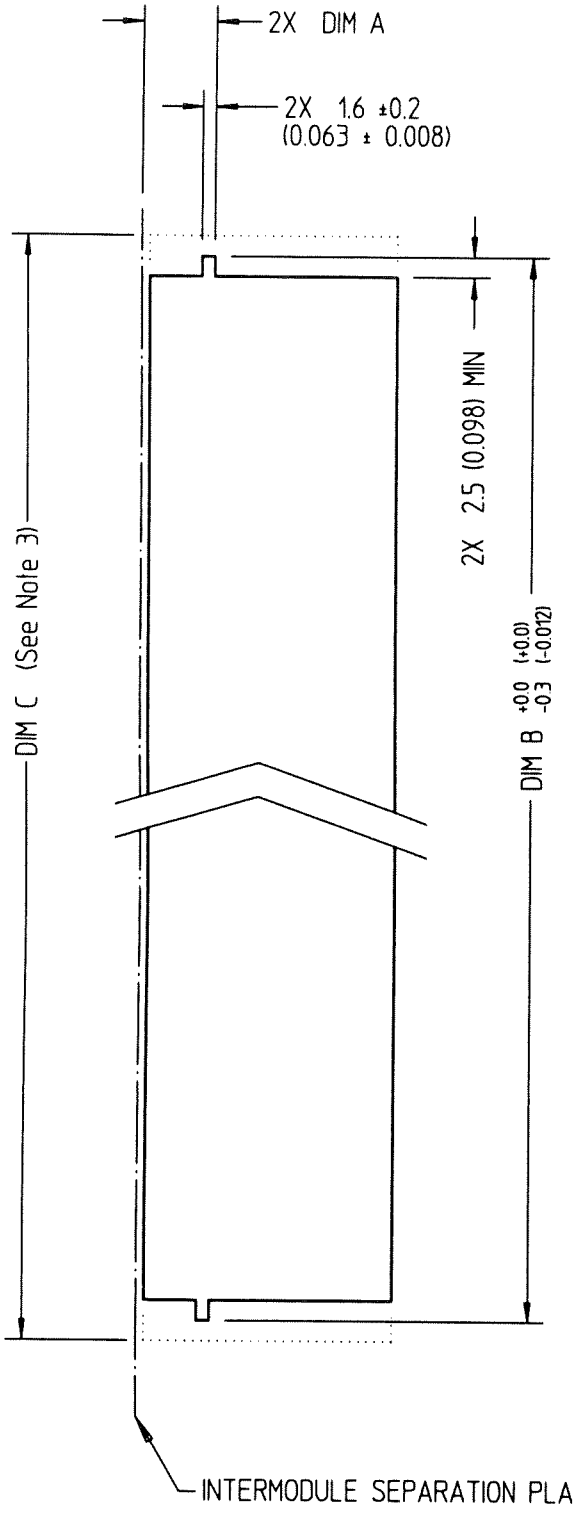


Figure B.20. D-size board assembly connector positions.

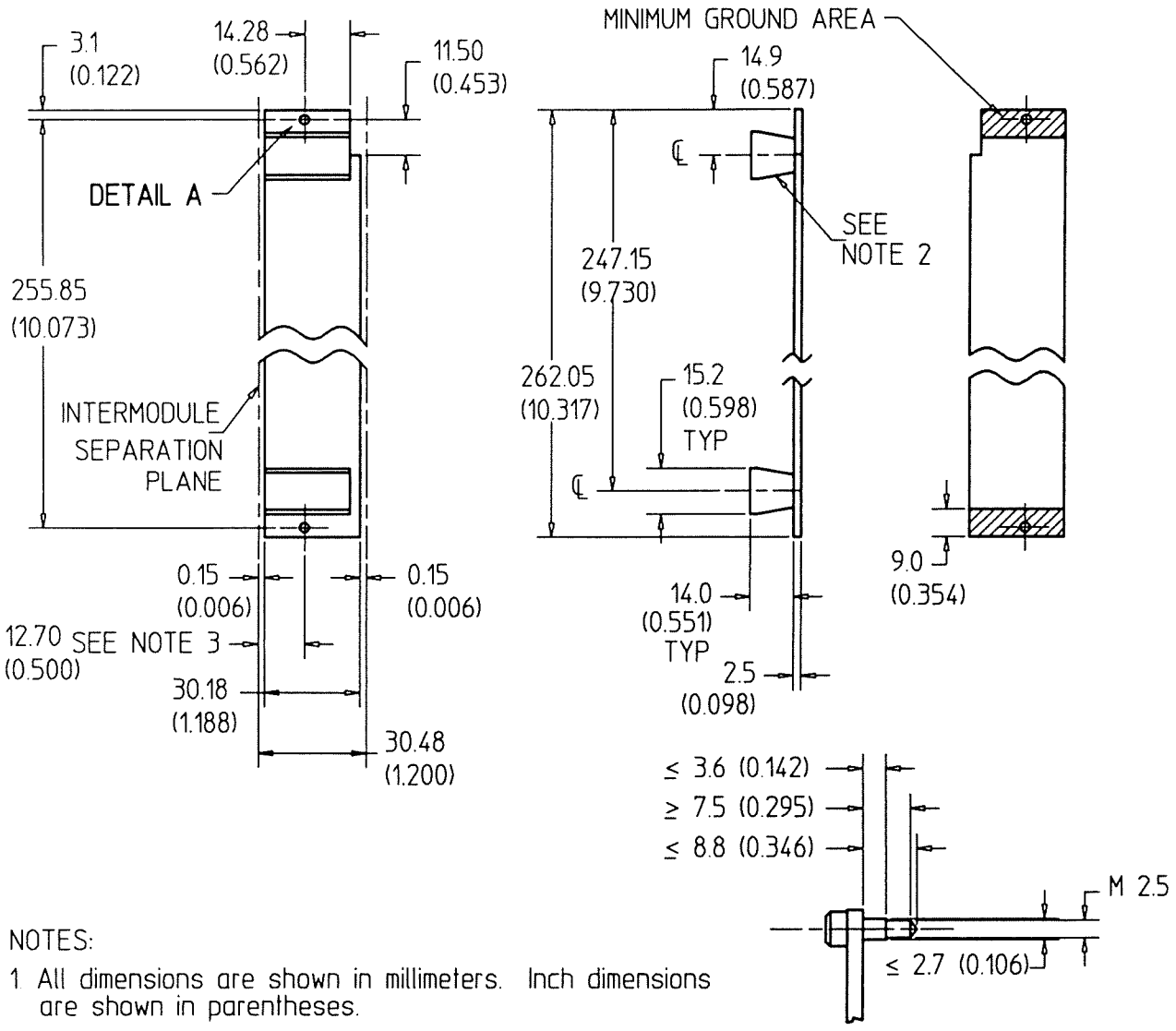


MODULE SIZE	DIM A	DIM B	DIM C
A	4.07 (0.160)	100.0 (3.927)	105.3 (4.146)
B	4.07 (0.160)	233.35 (9.187)	238.65 (9.396)
C	9.15 (0.360)	233.35 (9.187)	238.65 (9.396)
D	9.15 (0.360)	366.70 (14.437)	372.0 (14.646)

NOTES:

1. All dimensions shown in millimeters. Inch dimensions are shown in parentheses.
2. Refer to the figure "Module Envelope, Top View", View A-A.
3. Restriction imposed by injector surfaces. Refer to the section "Module Shielding", and to the figure "Module Injection Surface".

Figure B.21. Module edge guide feature.



NOTES:

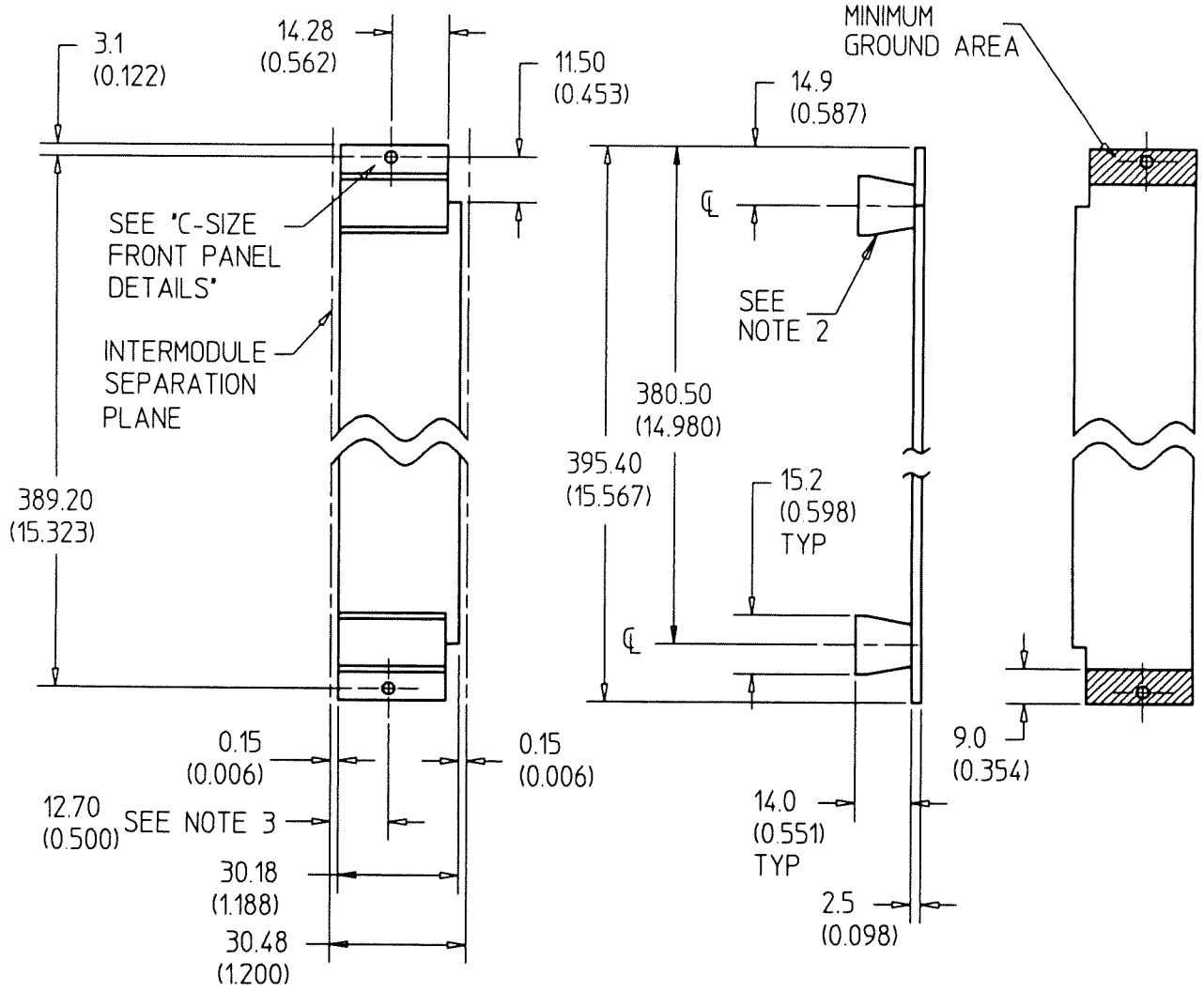
1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. Dimensions given for height, depth and position of handles are suggestions only.
3. RECOMMENDATION

   Locate the mounting hole 12.70 mm (0.500 in) from the intermodule separation plane.

   PERMISSION

   The mounting hole may be located 17.78 mm (0.700 in) from the intermodule separation plane.

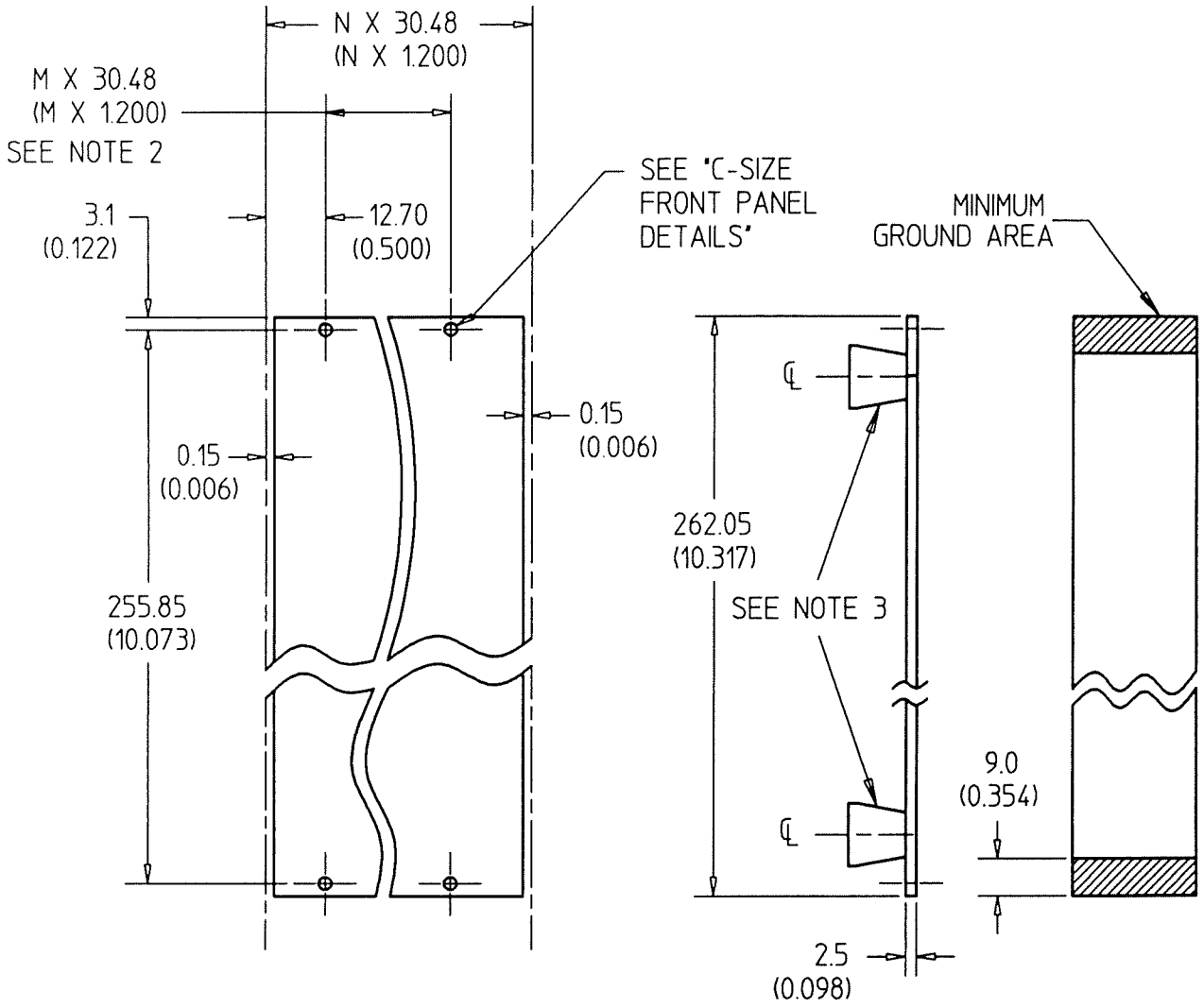
Figure B.22. C-size Front panel details.



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. Dimensions given for height, depth and position of handles are suggestions only.
3. RECOMMENDATION
   Locate the mounting hole 12.70 mm (0.500 in) from the intermodule separation plane.
   PERMISSION
   The mounting hole MAY be located 17.78 mm (0.700 in) from the intermodule separation plane.

Figure B.23. D-size Front panel details.



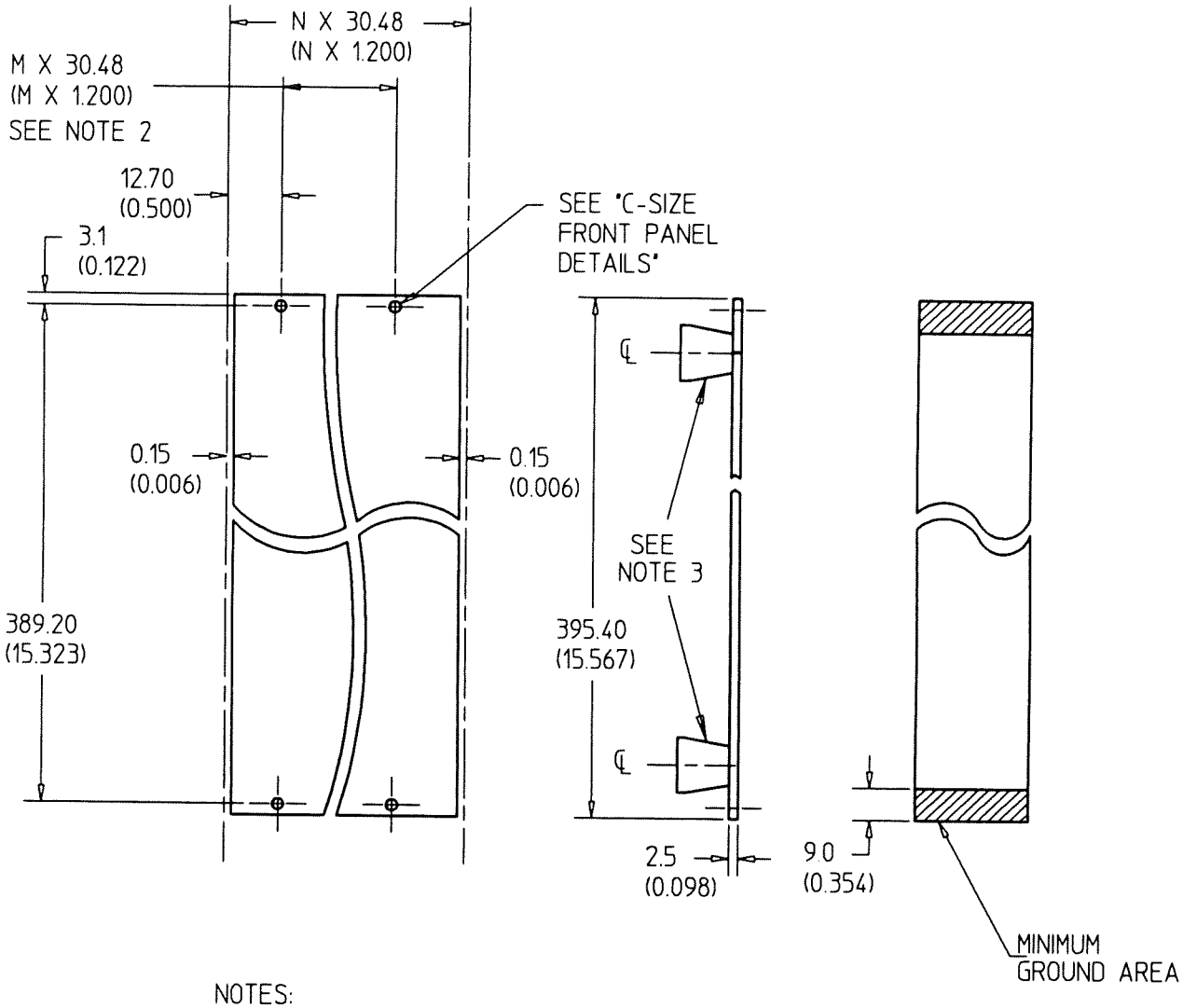
NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. RECOMMENTATION

   Where a panel is more than 30.48 mm (1.200 in) wide, use at least 4 mounting holes; two at the top and two at the bottom.
3. SUGGESTION

   IF handles are used on filler panels,
   THEN they should be positioned as shown in front panel detail.

Figure B.24. C-size filler panel.



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. RECOMMENTATION
   Where a panel is more than 30.48 mm (1.200 in) wide, use at least 4 mounting holes; two at the top and two at the bottom.
3. SUGGESTION
   IF handles are used on filler panels,
   THEN they should be positioned as shown in front panel detail

Figure B.25. D-size filler panel.

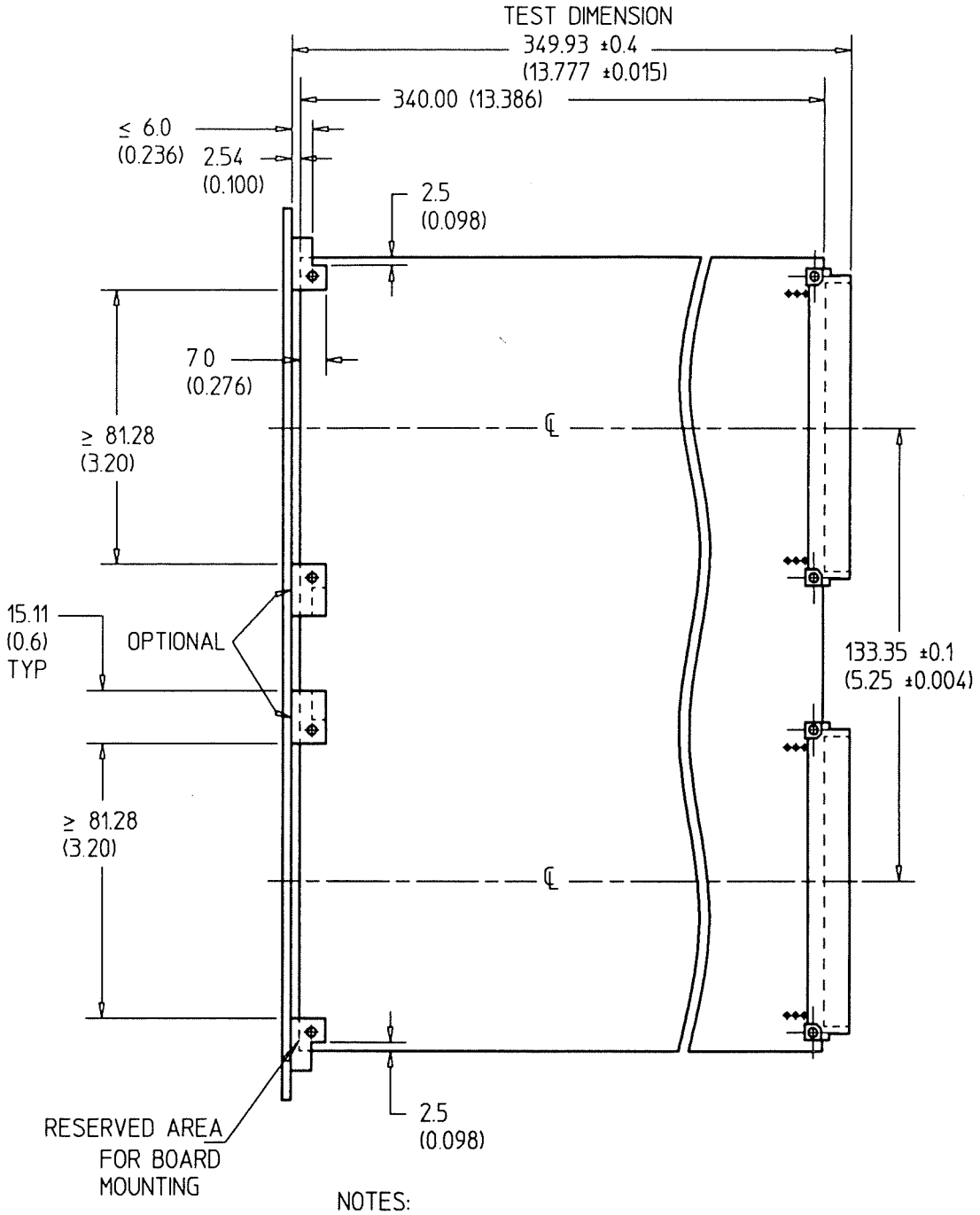
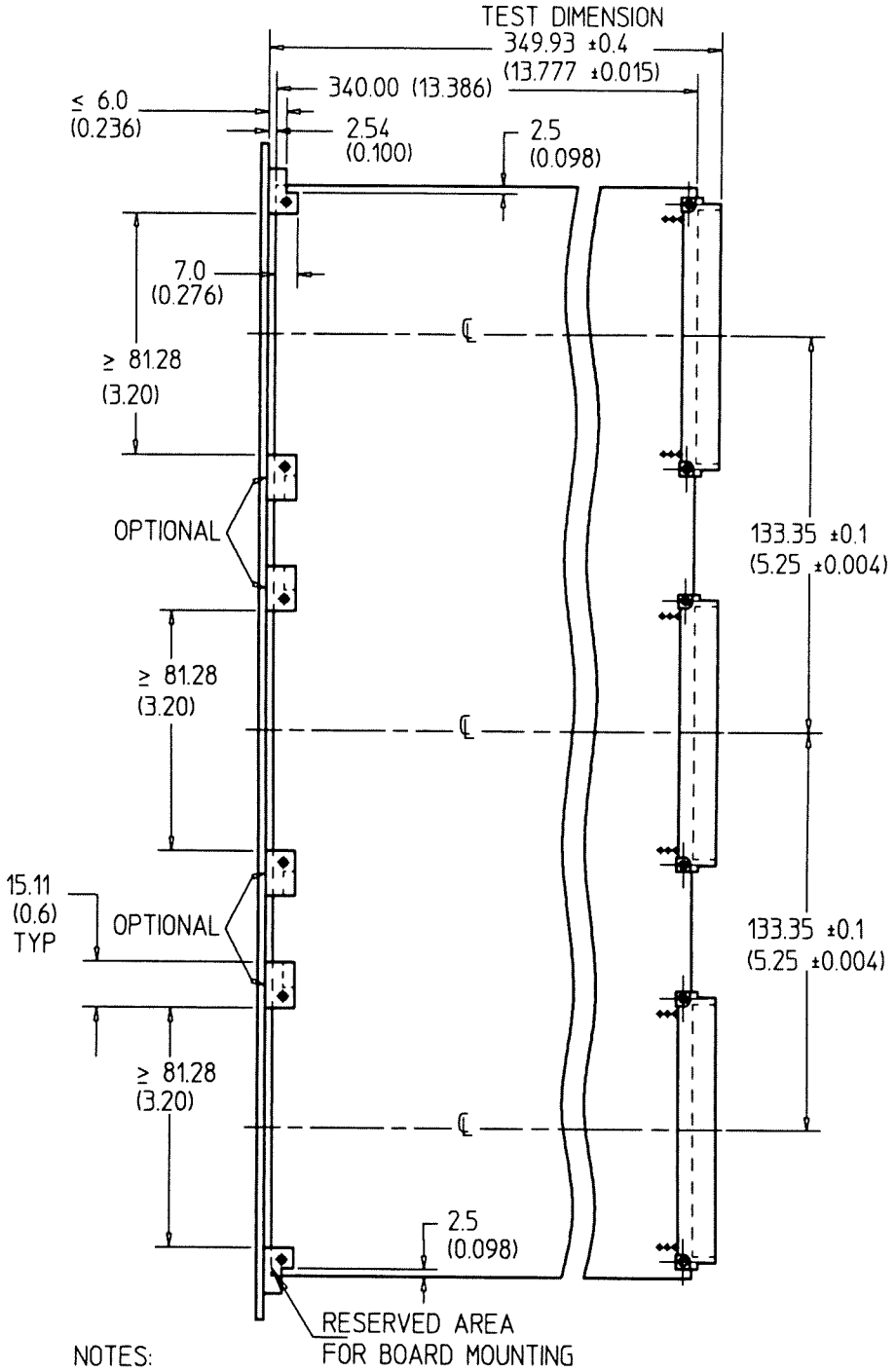


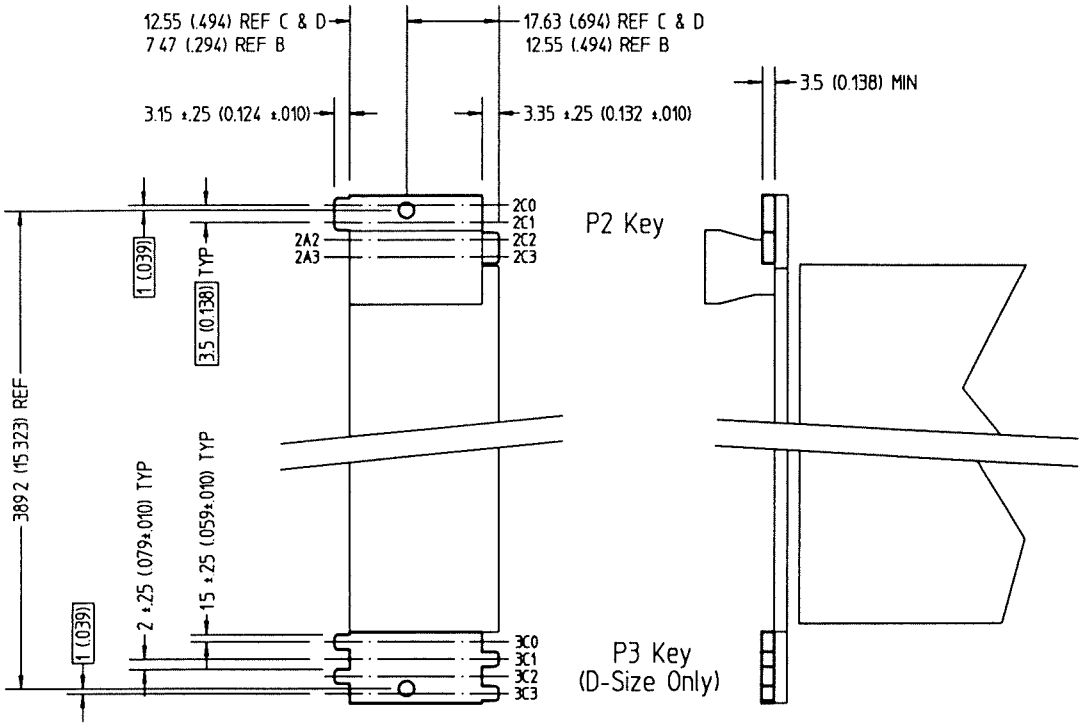
Figure B.26. Typical C-size front panel mounting and dimensions.



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. Dimensions for front panel mounting brackets are suggestions only. Mounting methods may vary.

Figure B.27. Typical D-size front panel mounting and dimensions.



NOTES:
1. Dimensions are in mm. Inch dimensions are in parentheses.
2. The example shown is keyed for TTL on P2 and Analog Low on P3.

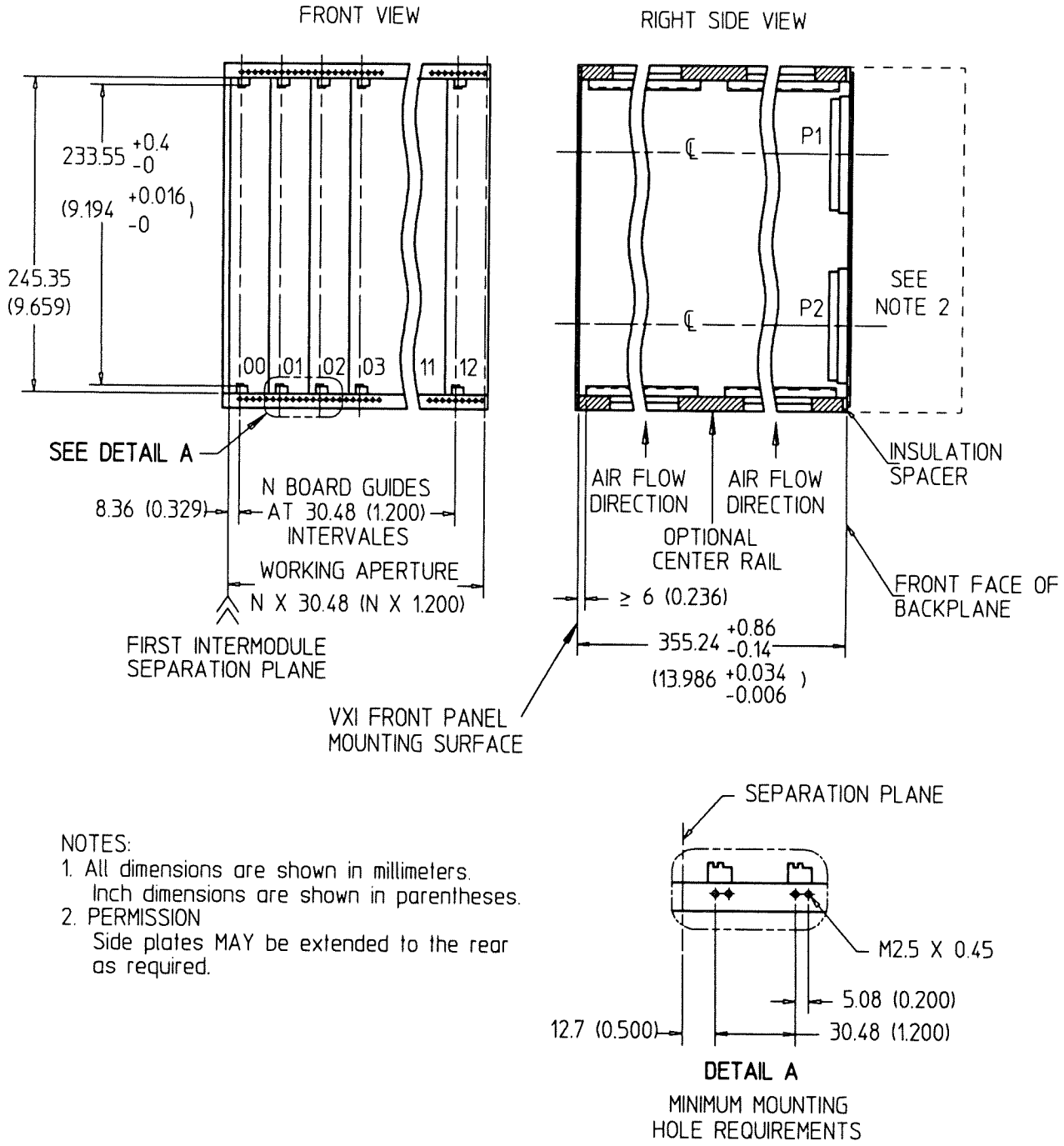
KEY NOMENCLATURE EXAMPLE: 2C3
- LOCAL BUS CONNECTOR, P2 OR P3
- SIDE OF MODULE, A OR C
- KEY NUMBER, 0 THRU 3

LOCAL BUS LOCKOUT KEY POSITIONS

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6			
	TTL	ECL	Analog Low	Analog Med	Analog High	Resrvd	No Local Bus	Sensor ±16V	Sensor ±42V
	A C	A C	A C	A C	A C	A C	A C	A C	A C
Key									
0	X -	- X	X -	X -	- X	- X	- -	X -	- -
1	X -	- X	- X	- X	X -	X -	- -	- X	- -
2	- X	X -	X -	- X	- X	X -	- -	- -	- -
3	- X	X -	- X	X -	X -	- X	- -	- -	- -

X Key Present

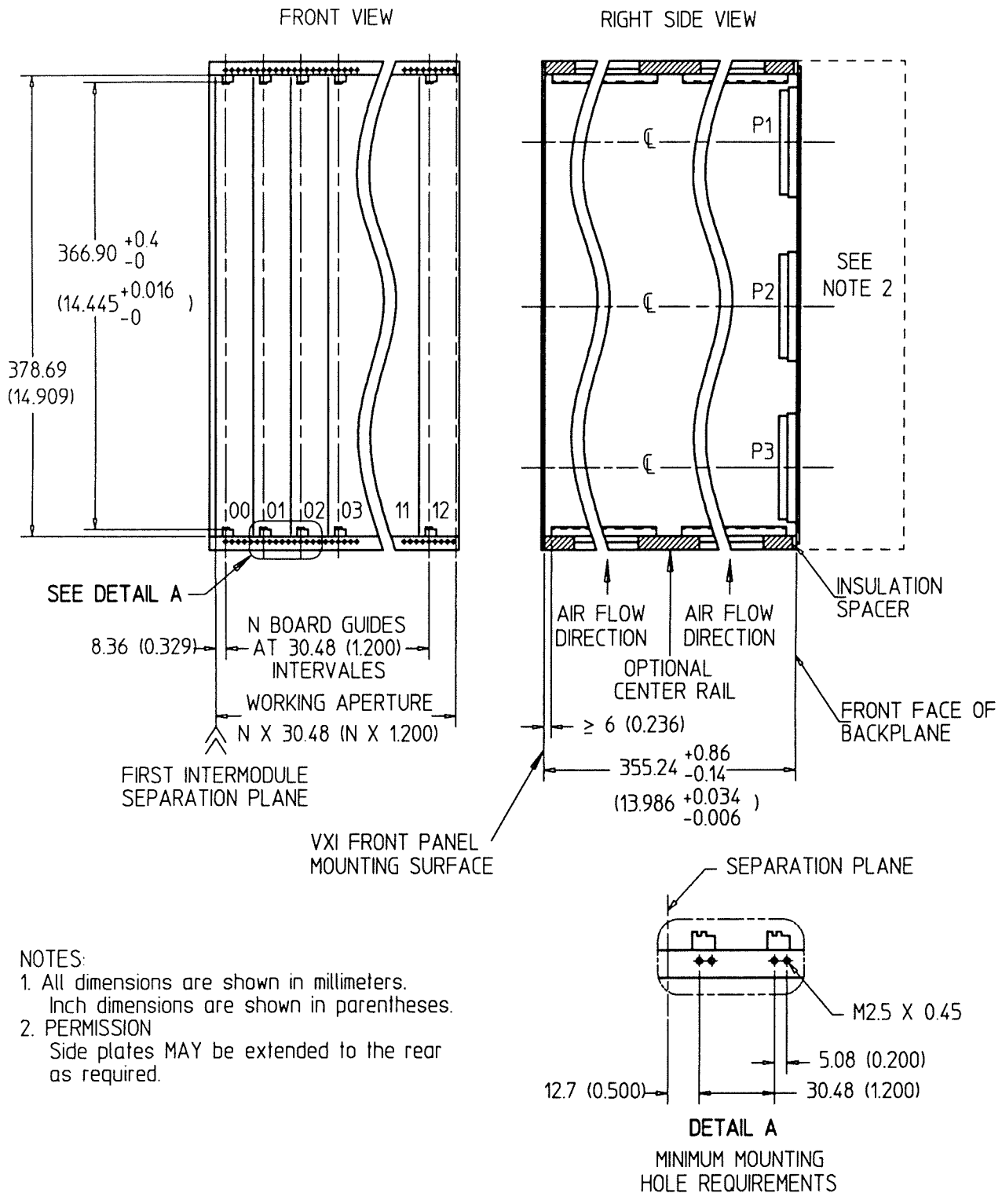
Figure B.28. Local bus lockout key details.



NOTES:
1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. PERMISSION
   Side plates MAY be extended to the rear as required.

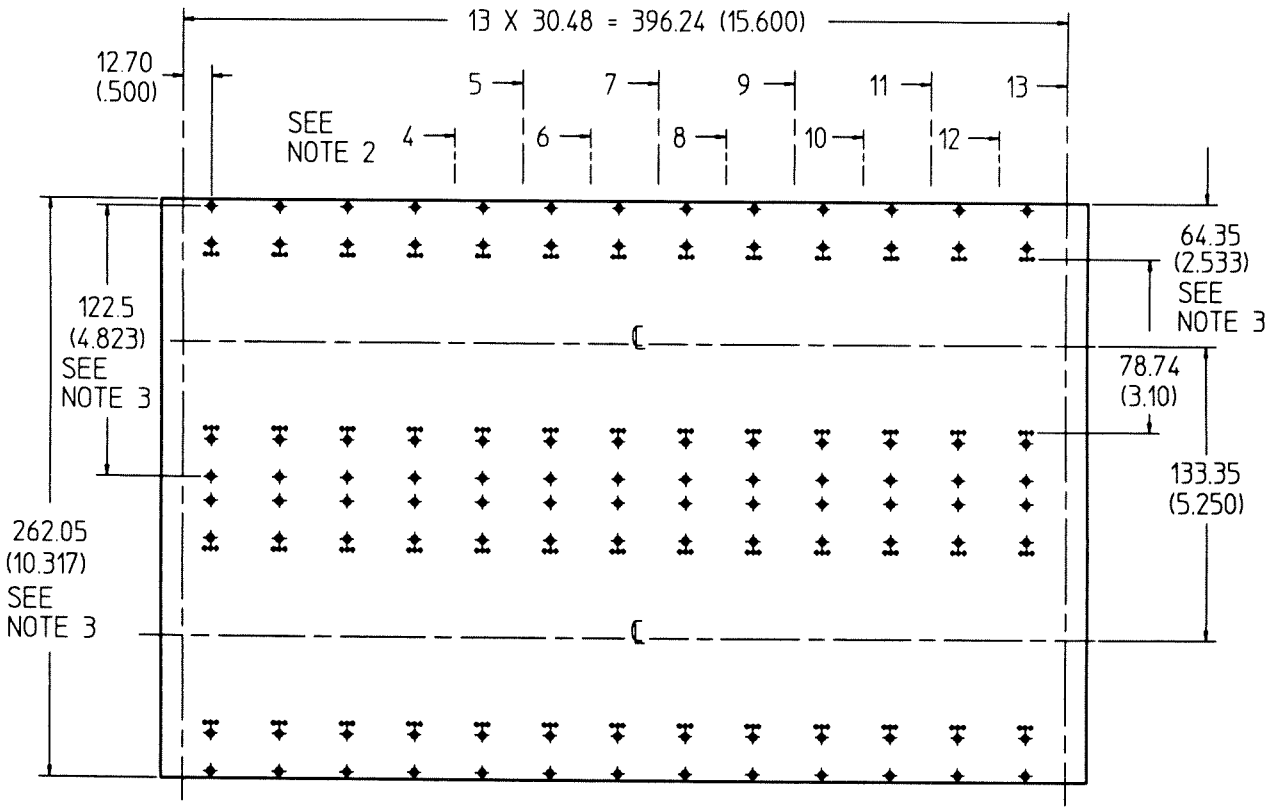
DETAIL A
MINIMUM MOUNTING HOLE REQUIREMENTS

Figure B.29. C-size mainframe drawing



NOTES:
1. All dimensions are shown in millimeters.
   Inch dimensions are shown in parentheses.
2. PERMISSION
   Side plates MAY be extended to the rear as required.

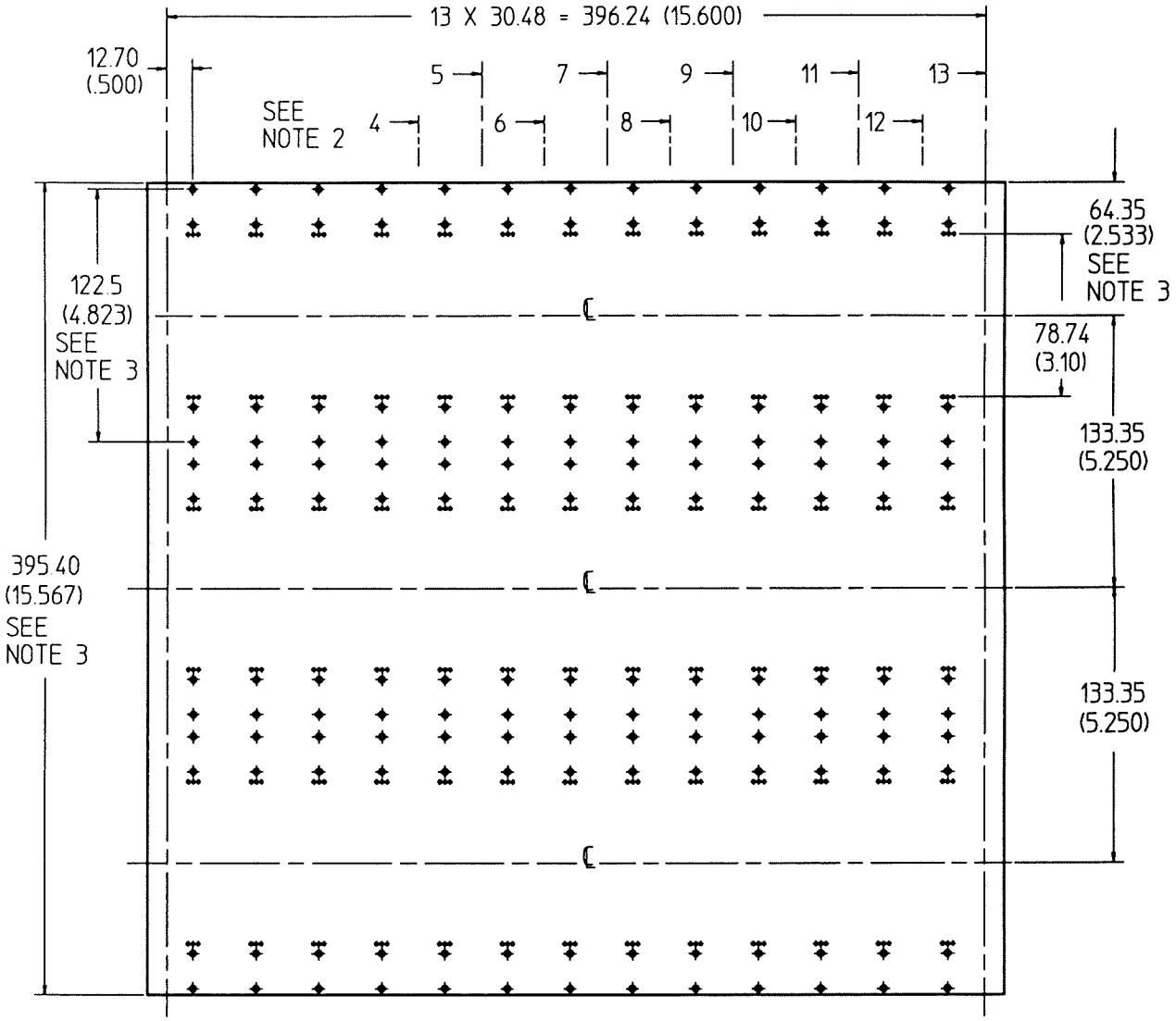
Figure B.30. D-size mainframe drawing



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. Backplane width varies depending on the number of slots.
3. These dimensions are shown as suggestions only. (See Backplane section)

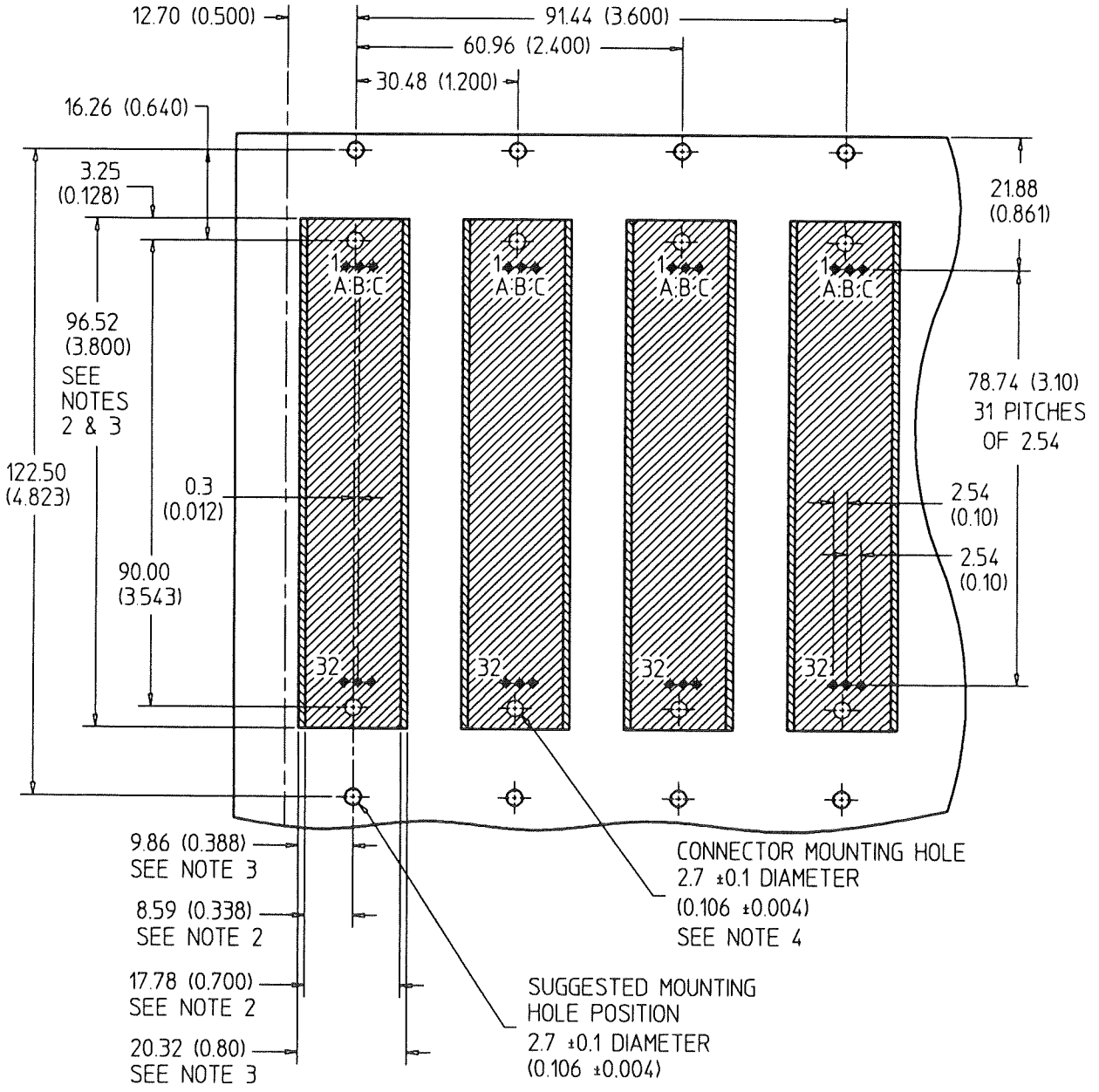
Figure B.31. C-size backplane drawing.



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. Backplane width varies depending on the number of slots.
3. These dimensions are shown as suggestions only. (See Backplane section)

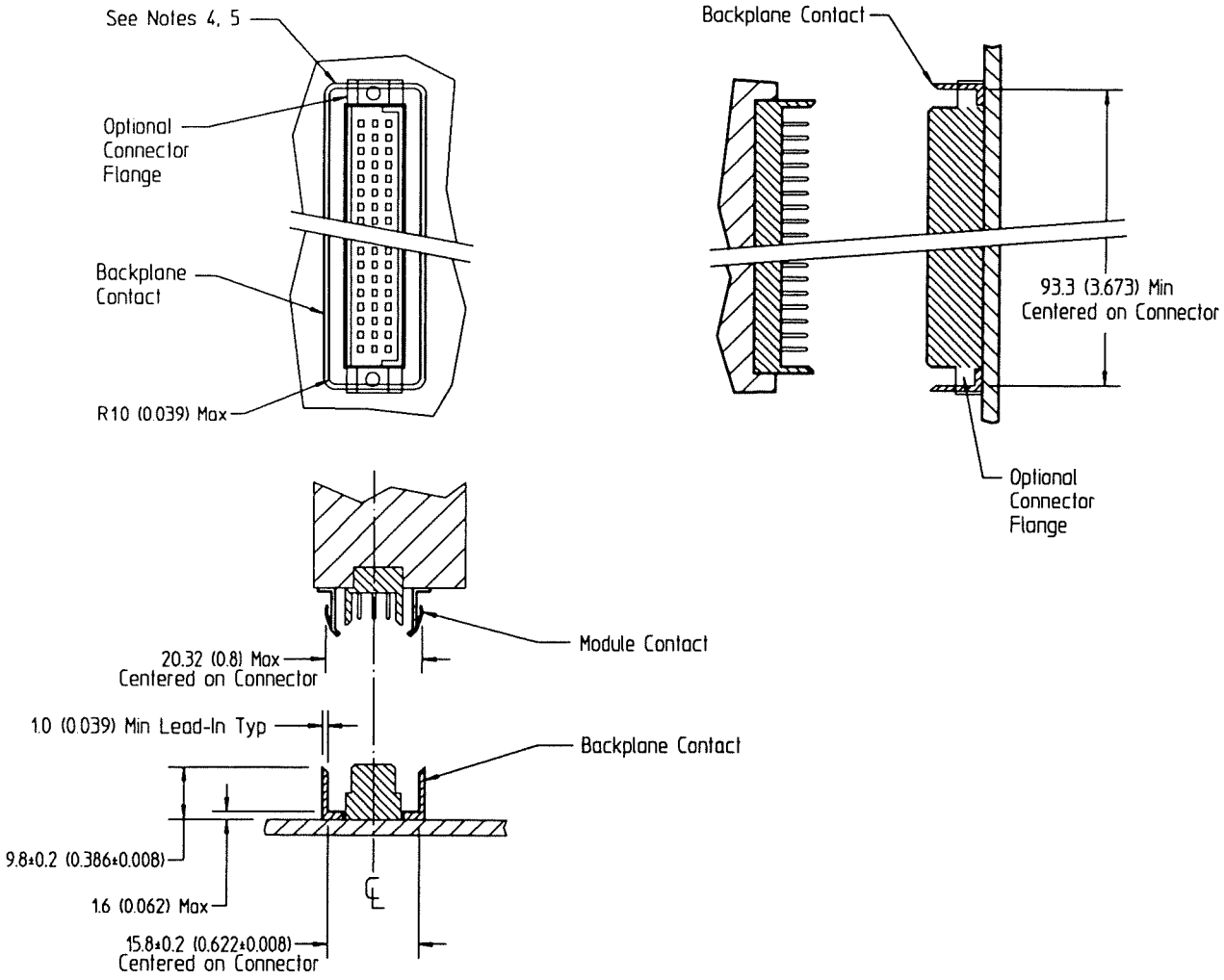
Figure B.32. D-size backplane drawing.



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. Minimim area of ground plane for module shield grounding ring around all J1, J2 and J3 connectors. (See MAINFRAME SHIELDING section)
3. No components this area except 96-pin connector.
4. Connector mounting hole may be required by connector shield, etc.

Figure B.33. Detailed backplane dimensions.



NOTES:

1. All dimensions are shown in millimeters. Inch dimensions are shown in parentheses.
2. RULE: Module contacts SHALL be compatible with the backplane contact dimensions shown.
3. RULE: Module contact design SHALL NOT impede mating of the backplane connectors. Sufficient spring contact flexure and lead-in is required.
4. PERMISSION: Backplane contacts MAY or MAY NOT extend around the end of the backplane connector as shown. Note that board guides prevent module contacts from extending around the end.
5. RULE: All module contacts SHALL be compatible with backplane contacts that extend around the end of the backplane connector.
6. RULE: Contact materials SHALL be galvanically compatible with Tin/Lead.
7. RULE: Contact life SHALL be consistent with the module and backplane connectors.
8. RULE: Connector shields SHALL NOT increase insertion force by greater than 50% above the module and backplane connectors.
9. PERMISSION: Module contacts and backplane contacts MAY or MAY NOT be removable.
10. OBSERVATION: Removable contacts allow the user to eliminate the connector shield. This may be useful for some applications. Example: Circuit ground may be isolated from chassis ground. Module manufacturers should evaluate the applicability of removable contacts for each module.
11. SUGGESTION: Mount backplane contacts to the backplane connector mounting holes.
12. OBSERVATION Primary function is rfi shielding, not low impedence ground path. Performance will vary with implementation.

Figure B.34. Backplane connector shield.

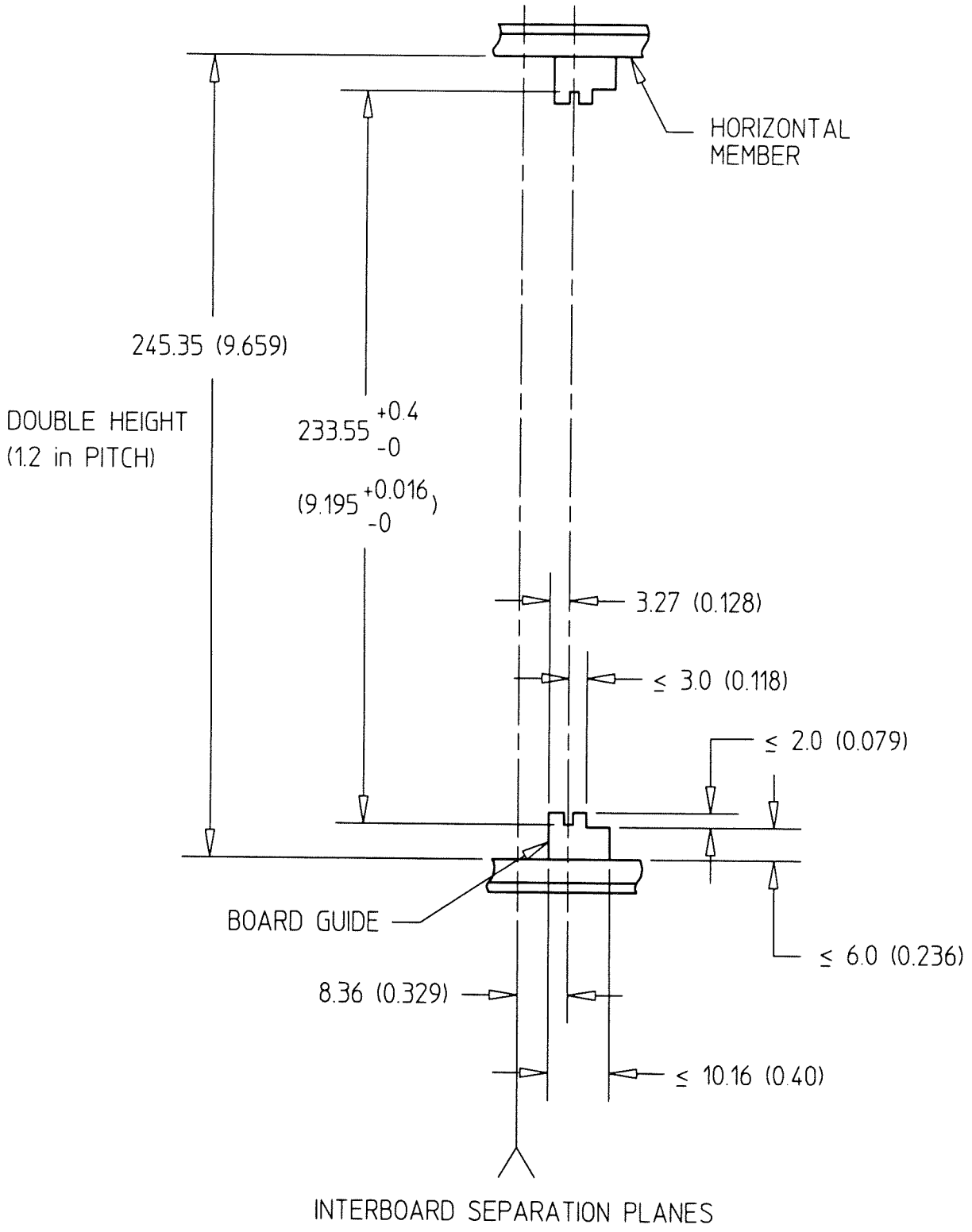


Figure B.35. C-size board guide detail

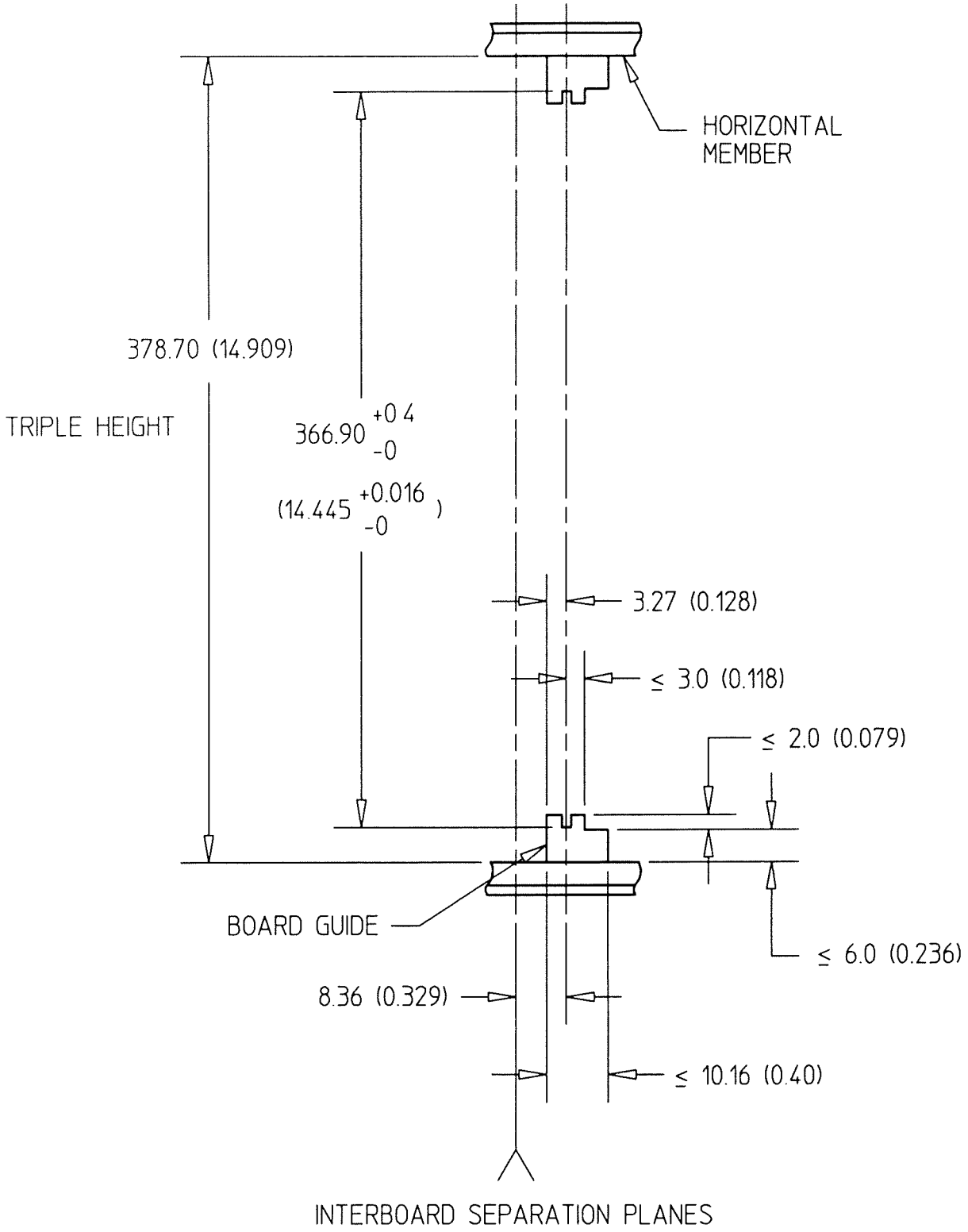


Figure B.36. D-size board guide detail

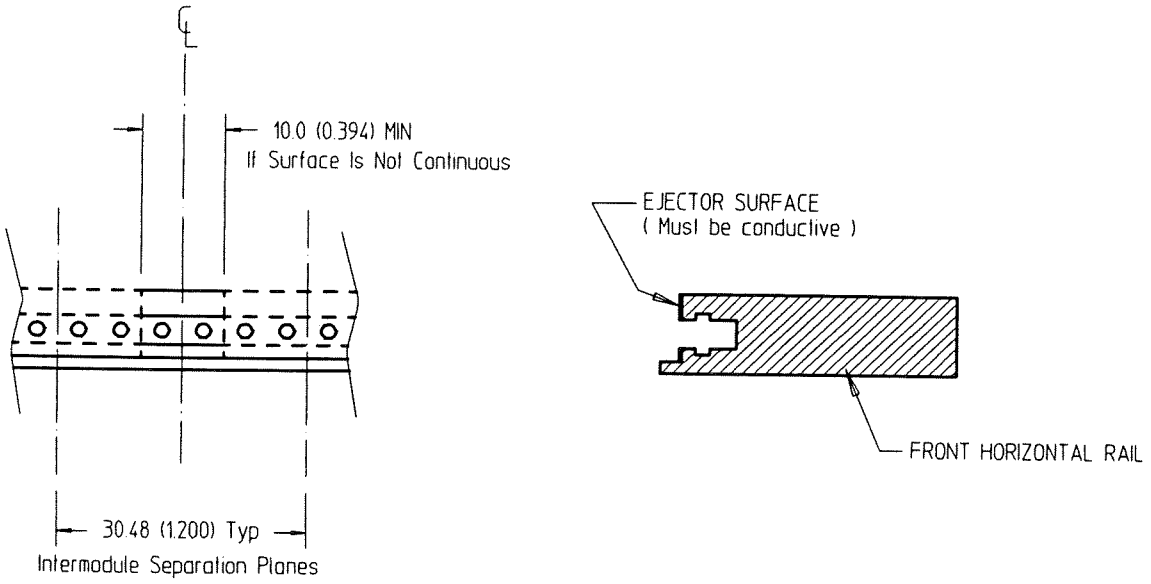


Figure B.37. Module extraction surface.

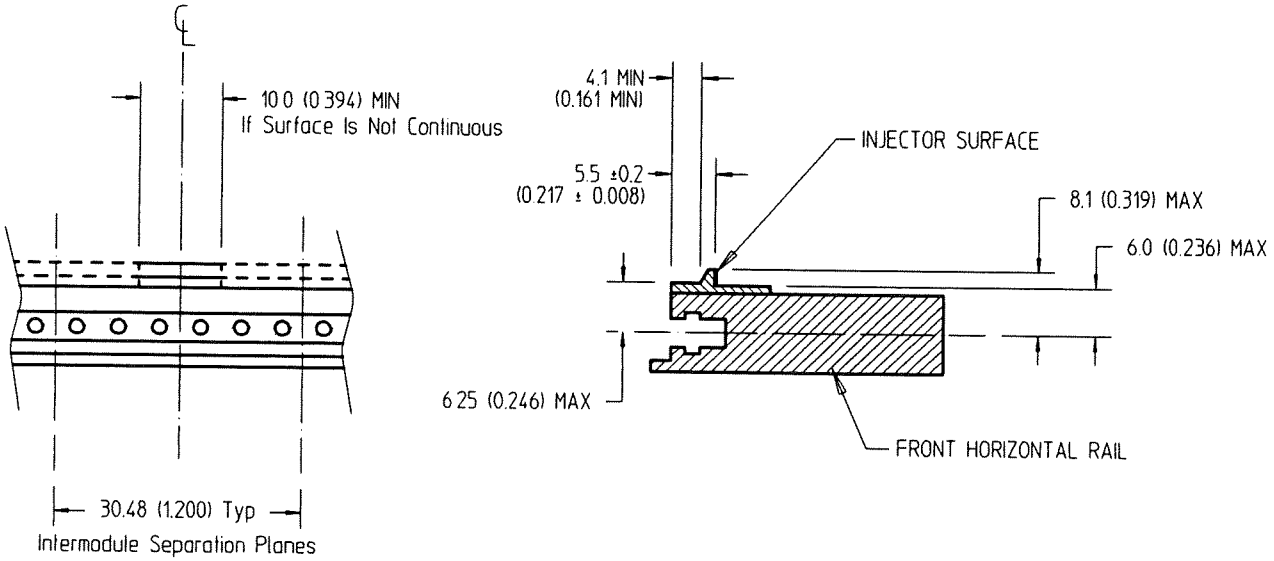
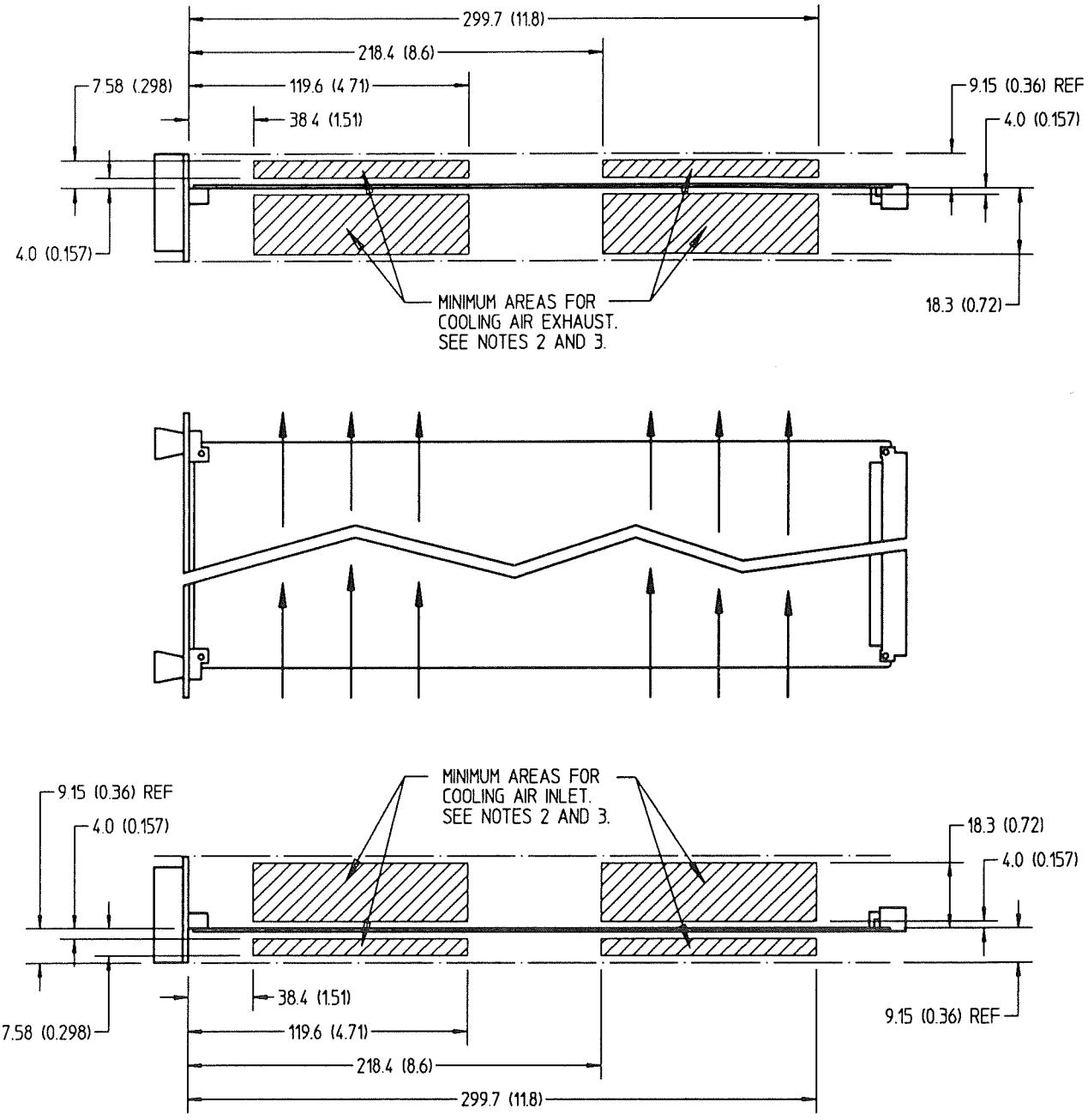


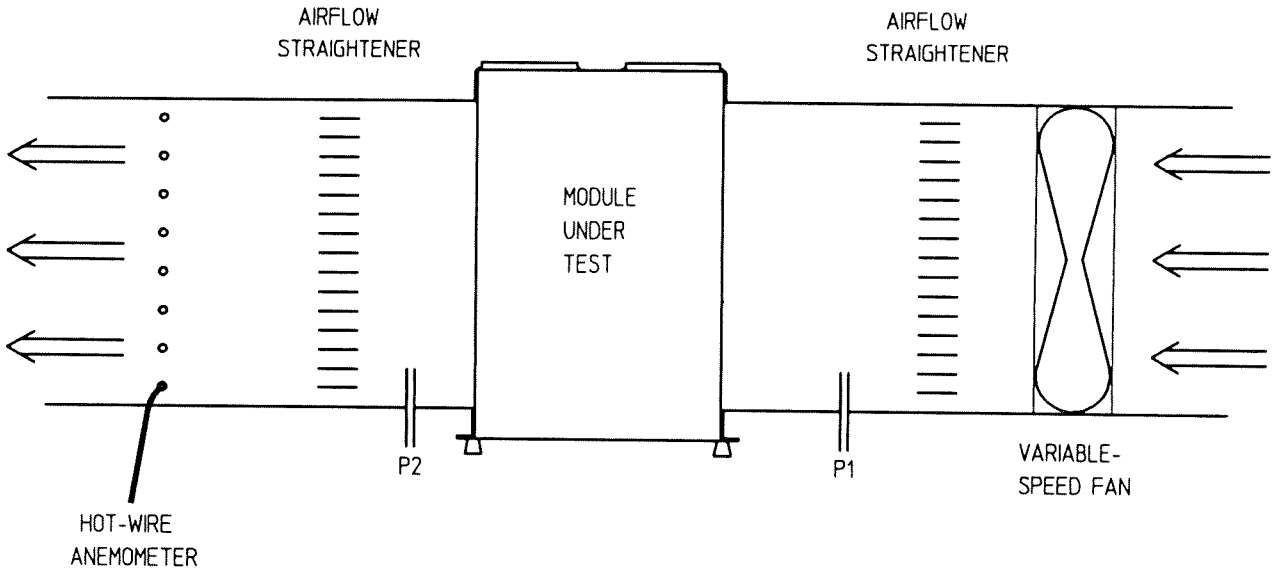
Figure B.38. Module injection surface.



NOTES:

1. All dimensions in millimeters. Inch dimensions in parentheses.
2. Minimum areas apply to mainframes only. Module designers should use them as a guideline. For example, many modules may not require cooling air on the solder side.
3. Minimum areas may or may not be completely open. They may consist of screens, filters, hole patterns, etc.

Figure B.39. C and D size module inlet and exhaust areas.

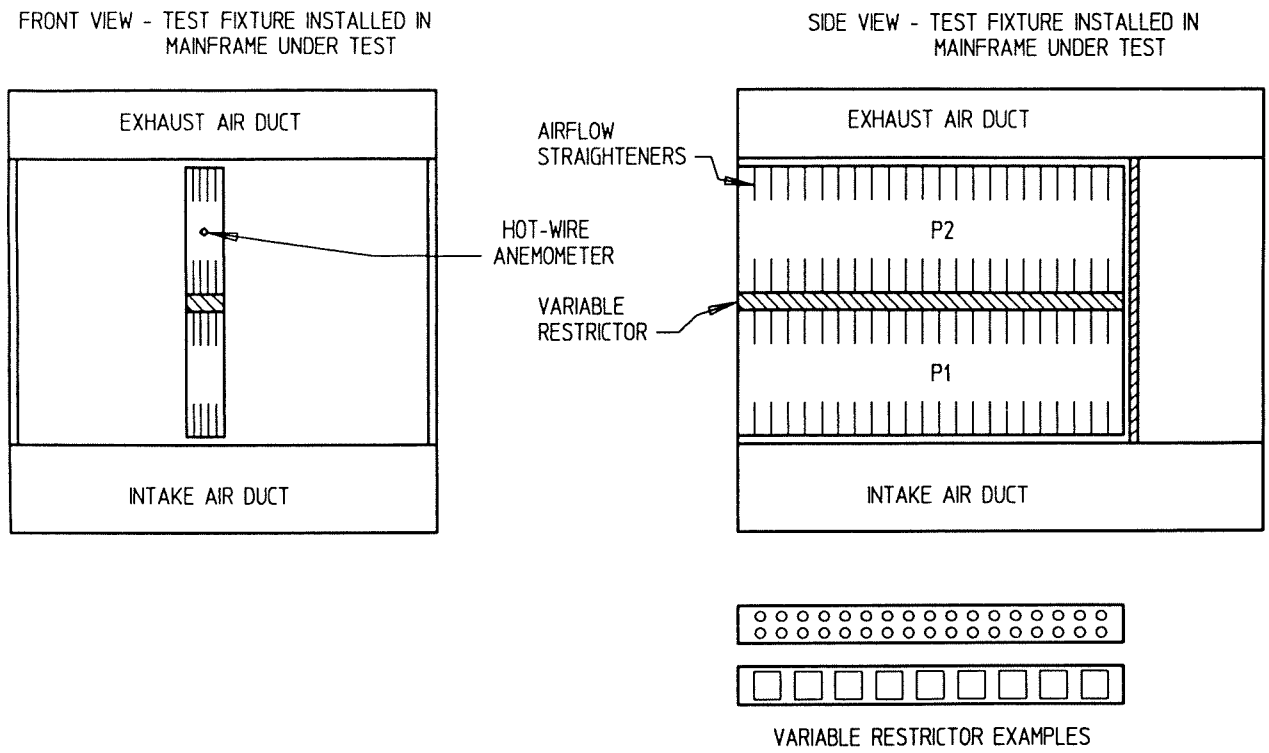


PROCEDURE

1. Build a test fixture as shown. The inlet and exhaust areas of the fixture should match those specified in the figure 'C and D size module inlet and exhaust areas'.
2. Turn fan on low.
3. Allow system to reach steady-state.
4. Measure P1 and P2. P = P1 - P2
5. Determine airflow (G) by sampling several points in the duct with a hot-wire anemometer.*
6. Plot point (G,P).
7. Increase the fan speed and repeat steps 3-7 until the relationship between P and G is sufficiently defined.
8. Establish the minimum airflow requirements of the module based on power consumption, reliability needs, and the intended application.
9. The operating point is defined as the intersection between the minimum airflow required and P v.s. G. See the section on module cooling.

* Note: Great care is required to reduce the effects of boundary layers, turbulence, flow direction, etc. For more detailed information on airflow testing, refer to published standards such as ANSI/AMCA 210-85, etc.

Figure B.40. Simplified test for module cooling specifications.

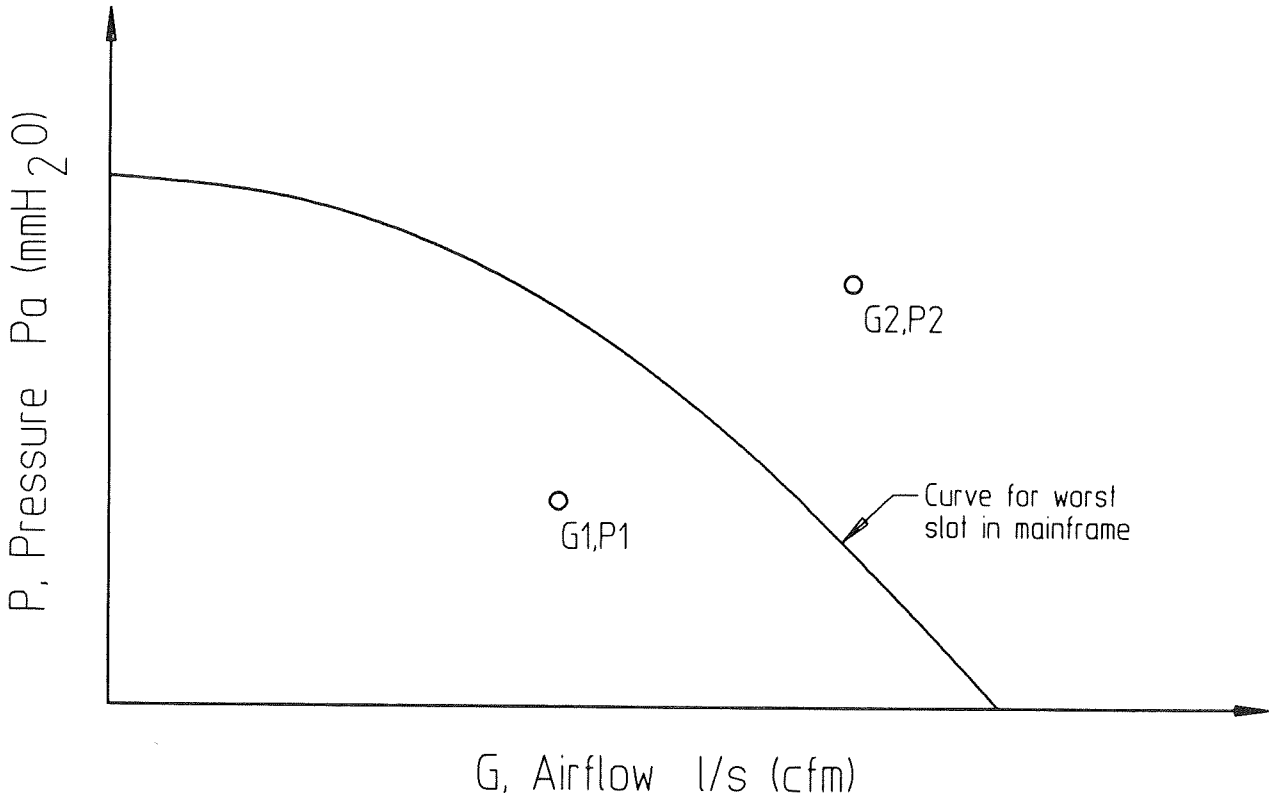


PROCEDURE

1. Build a test fixture as shown, same size as a module. The inlet and exhaust areas of the test fixture should not restrict the flow of air within the module envelope. See figure "Module envelope, top view".
2. With all other slots empty, place the fixture in a single slot. Remove filler panels from all empty slots.
3. Insert the variable restrictor with least resistance to airflow. Turn on the mainframe.
4. Allow system to reach steady-state.
5. Measure P1 and P2. P = P1 - P2.
6. Determine airflow (G) by sampling several points in the fixture with a hot-wire anemometer.*
7. Plot point (G,P).
8. Change the airflow restrictor, slightly increasing the resistance to airflow. Repeat steps 4-8 until the relationship between P and G is sufficiently defined.
9. Repeat steps 2-8 for other slots until the "worst slot" is identified.
10. It may be necessary to specify more than one curve (P v.s. G) if the mainframe has features such as variable speed fans, large airflow differences between slots, etc.

* Note: Great care is required to reduce the effects of boundary layers, turbulence, flow direction, etc. For more detailed information on airflow testing, refer to published standards such as ANSI/AMCA 210-85, etc.

Figure B.41. Simplified test for mainframe cooling specifications.



IF the module operating point (G1,P1) falls BELOW the worst slot curve, THEN the module WILL receive adequate cooling.

IF the module operating point (G2,P2) falls ABOVE the worst slot curve, THEN the module WILL NOT receive adequate cooling.

Note: IF the module operating point (G,P) falls on or near the worst slot curve, THEN more detailed analysis may be required to determine whether cooling is adequate.

Figure B.42. Example of ΔP versus airflow for system integration.

B.8 EMC and SYSTEM POWER

B.8.1 Introduction

The VXIbus backplane is governed by the following rules.

RULE B.8.1:
VXIbus subsystems **SHALL** provide power conductors for distribution of +5V, +5 STDBY, +12V, -12V, +24V, -24V, -5.2V, and -2V to all of the power pins specified in the pin assignments.

B.8.2 Power Distribution

Power in a VXIbus system is distributed on the backplane as regulated direct current (DC) voltages. The available voltages are:

+5 V dc	This is the main power source for most VXIbus systems. All systems require this voltage for bus communications.
+/-12 V dc	These are often used for powering analog devices, disc drives, and communication interfaces.
+/-24 V dc	These are used for powering analog signal sources, and to derive other voltage levels (e.g. +/-15 V dc) as needed using on-board regulators.
-5.2 V dc	This is used for ECL devices.
-2 V dc	This is used for termination of ECL loads.
+5 V dc STDBY	This is used to sustain memory, clocks, etc. when the +5 V dc power is lost.

B.8.3 Power Pins

RULE B.8.2:
The current flow per pin on any connector **SHALL** comply with Figure 5-7 in the VMEbus specification based on temperature rise considerations in the target system.

OBSERVATION B.8.1:
A 1 Ampere per pin limit allows the VMEbus rated connector to operate at 55°C ambient.

B.8.4 DC Voltage Specifications

RULE B.8.3:
Any voltage furnished by the mainframe power supply **SHALL** comply with the maximum Allowed Variation and maximum allowed DC Load Ripple/Noise in Table B.7, up to the mainframe's maximum rated current (I_{MP}).

VOLTAGE	DESCRIPTION	ALLOWED VARIATION	DC LOAD RIPPLE/NOISE	INDUCED RIPPLE/NOISE
+5V	+5 V dc	+0.25V/-0.125V	50 mV	50 mV
+12V	+12 V dc	+0.60V/-0.36V	50 mV	50 mV
-12V	-12 V dc	-0.60V/+0.36V	50 mV	50 mV
+24V	+24 V dc	+1.20V/-.72	150 mV	150 mV
-24V	-24 V dc	-1.20V/+.72	150 mV	150 mV
-5.2V	-5.2 V dc	-0.260V/+0.156V	50 mV	50 mV
-2V	-2 V dc	-0.10V/+0.10V	50 mV	50 mV
+5V STDBY	+5 V dc standby	+0.25V/-0.125V	50 mV	50 mV

TABLE B.7. Power Supply Voltage Specifications

DC Load Ripple/Noise is the maximum Periodic And Random Deviations (PARD) generated by the power supply under all DC load conditions up to the Mainframe Peak Current, measured as the peak to peak voltage in the DC to 10 MHz bandwidth. This is the V_{ML} parameter in Figure B.44 and Figure B.46.

Induced Ripple/Noise is the additional peak to peak ripple that may exist on the backplane power supply pins due to injected current from operating modules. This is the V_{MI} parameter in Figure B.44 and Figure B.46.

Mainframe Peak Current (I_{MP}) is the rated DC current of a particular voltage supplied by the mainframe.

Mainframe Dynamic Current (I_{MD}) is the rated dynamic current capacity of a particular voltage supplied by the mainframe for the frequencies from 20 Hz to 1 GHz.

RULE B.8.4:
Power Supplies source power to the modules and **SHALL NOT** be used to sink power other than transient current from protection circuits.

OBSERVATION B.8.2:
The non-symmetric variation given in Table B.7 ensures that the DC power remains within the tolerance required by most ICs despite the typical voltage drops that occur in the power distribution network. Placing the power supply sense point near the power input point prevents boards near the power input point from receiving too high a voltage.

RULE B.8.5:
All voltages which are supplied by a mainframe **SHALL** tolerate peak to peak current variations of I_{MP} from DC to 20 Hz without generating peak to peak voltage variations in the DC to 20 Hz frequency range exceeding the Induced Ripple/Noise limit of Table B.7.

RULE B.8.6:
All voltages which are supplied by a mainframe **SHALL** tolerate peak to peak current variations as specified by the Max. Rated Mainframe Dynamic Current (I_{MD}) in Figure B.43 without generating total (Induced and Load Ripple/Noise) peak to peak voltage variations exceeding the limits specified in Figure B.44. For the test method see Section B.8.7.2 "Induced Ripple Noise Test of Mainframes". I_{MD} is the rated dynamic current of the mainframe supply. V_{MI} is the peak to peak Induced Ripple/Noise specified in Table B.7. V_{ML} is the peak to peak DC Load Ripple/Noise specified in Table B.7. *Slots* is the number of VXIbus module slots in the mainframe.

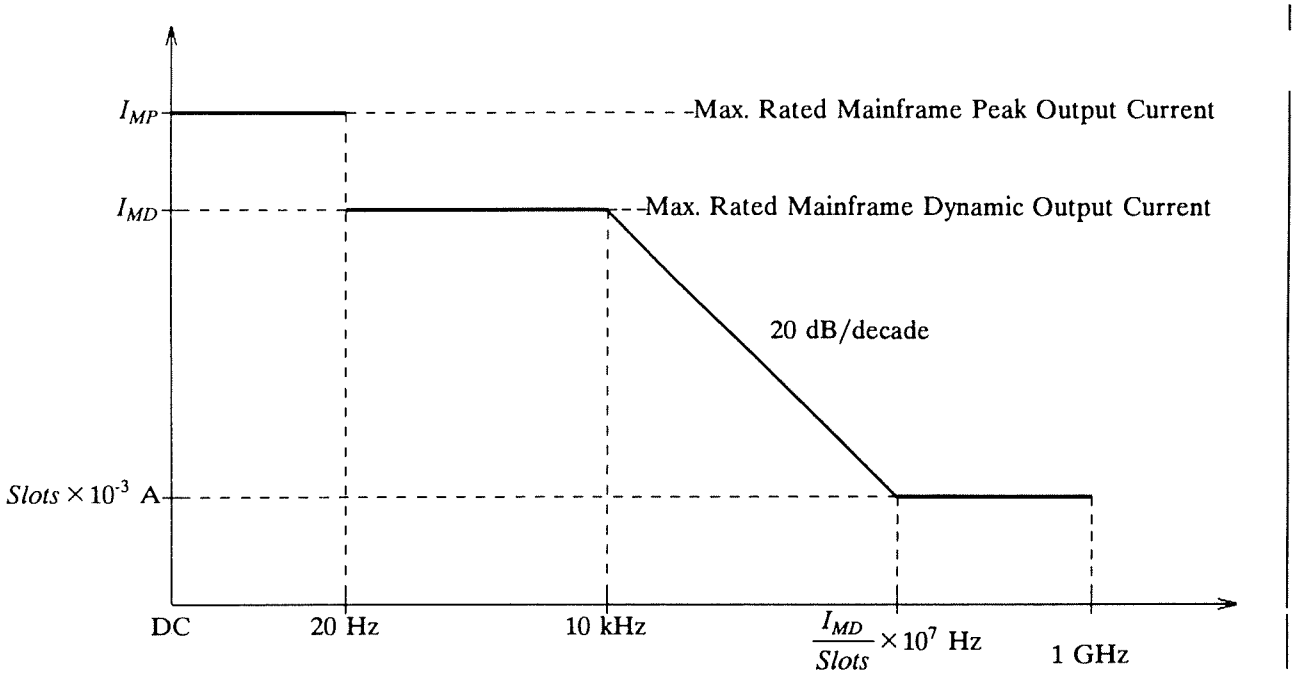


Figure B.43. Mainframe Load Current

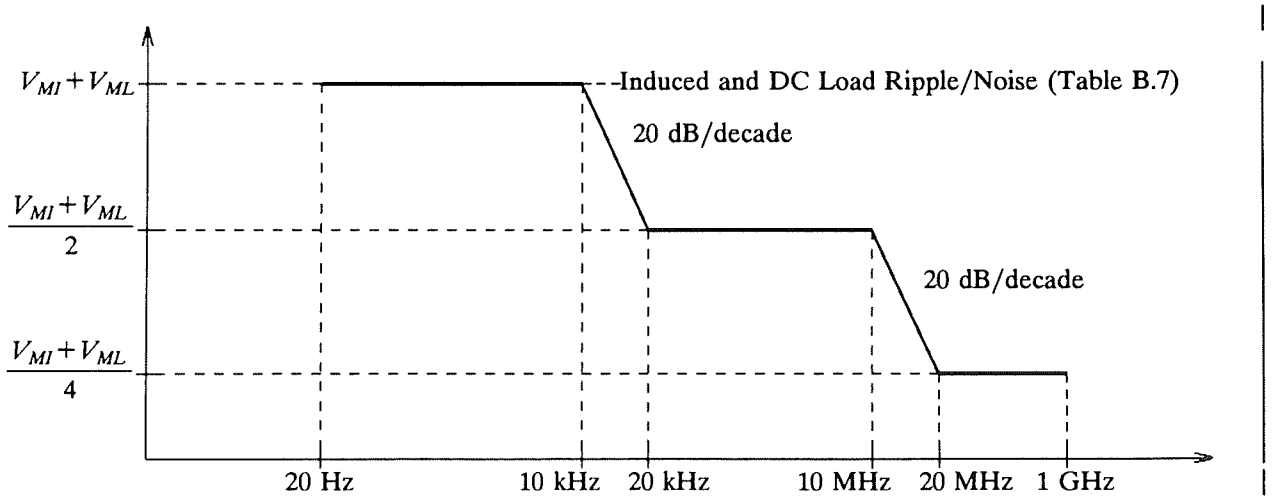


Figure B.44. Mainframe Induced and Load Ripple/Noise Voltage Limits

OBSERVATION B.8.3:

The rated Mainframe Dynamic Current (I_{MD}) is chosen by the manufacturer to ensure compliance with the above Induced Ripple/Noise specification.

B.8.5 Power Management

The power compatibility of mainframes and modules is accomplished by documenting the mainframe output power (both peak and dynamic) and module input power (both peak and dynamic). Mainframe power supplies will identify all voltage and current capabilities which are present in the mainframe. Modules will identify all voltage and input current requirements for proper operation. This documentation of the power requirements will allow the system integrator to verify compatibility between mainframe output power capability and module supply requirements.

Successful integration of a VXIbus instrument system depends on the power compatibility of mainframes and modules. The individual dynamic currents of modules will add together to form complex dynamic currents that the mainframe must tolerate in order to maintain its ripple specifications. The VXIbus rules establish a minimum level of mainframe and module documentation that the manufacturer must comply with.

Peak Module Current (I_{Pm}) is the maximum rated instantaneous current measured in the DC to 10 MHz bandwidth which is drawn by a module for a particular supply voltage during normal operation.

Dynamic Module Current (I_{Dm}) is the maximum rated dynamic current required by the module. I_{Dm} is chosen by the manufacturer so that the module meets the conducted emission test from 20 Hz to 1 GHz.

RULE B.8.7:

Mainframe Peak Current (I_{MP}) and Mainframe Dynamic Current (I_{MD}) **SHALL** be documented by the mainframe manufacturer for every voltage, even if not supported.

RECOMMENDATION B.8.1:

Mainframe should be marked with current rating of all voltages listed even if not supplied.
Example: +5V STDBY = 0.0 A Peak, 0.0 A Dynamic (if not supported).

RULE B.8.8:

Peak Module Current (I_{Pm}) and Dynamic Module Current (I_{Dm}) **SHALL** be documented by the module manufacturer for all power supply voltages used.

RECOMMENDATION B.8.2:

Modules should be marked with input current specifications for all voltages used.

OBSERVATION B.8.4:

The module peak current is measured with a 10 MHz bandwidth instrument to ensure that contributions from all potentially significant frequency components are included. For proper operation, the peak current capacity of the mainframe (I_{MP}) is required to exceed the sum of the Peak Module Currents (I_{Pm}) for each supply used. If the dynamic current capacity of the mainframe (I_{MD}) exceeds the sum of the dynamic current requirements of the modules (I_{Dm}), then proper power supply ripple performance is also guaranteed. However, if the dynamic currents of the modules exceed that of the mainframe, the specified ripple limits might be exceeded, and more detailed specifications are required from the mainframe and module manufacturer(s).

The final determination of compatibility may be application specific. The mainframe manufacturer may provide a curve that shows the mainframe dynamic current versus frequency. The module manufacturer may provide dynamic module input current versus frequency for particular applications. For example, a pulse generator that can drive 10 volts into 50 ohms could have curves for 50 ohms, 75 ohms, etc. The module manufacturer may also provide voltage ripple specifications for the module if it is more tolerant than the VXIbus specification defined in Table B.7.

OBSERVATION B.8.5:

The required current specifications of mainframes and modules reflect the products in an operational state after power up. Mainframes and modules might act differently during the power-up process.

RECOMMENDATION B.8.3:
Mainframes should handle large inrush currents at power up gracefully, without shutting down.

RECOMMENDATION B.8.4:
Design a module to present a reasonable load to the mainframe power supplies during the power up period. The DC current required by the module, when it is powered by any DC voltage between 0 and V_{supply}, should not exceed the current drawn from an equivalent resistive load of $\frac{V_{supply}}{I_{Pm}}$. Inrush current, due to a module's supply capacitance while the power supply voltage is increasing, is allowed to exceed this value. Limit the module's input capacitance to 200 μF or less on each supply voltage used. Design module turn on currents, such as may be required for ovens and disk drives, to comply with the module's peak and dynamic current specifications. These restrictions allow a VXIbus system to behave gracefully during the power-up period.

B.8.6 Electromagnetic Compatibility (EMC) of Modules

B.8.6.1 CONDUCTED EMISSIONS

RULE B.8.9:
The maximum instantaneous current drawn by a module **SHALL NOT** exceed the Peak Module Current (I_{Pm}) specified by its manufacturer for any power supply voltage used.

RULE B.8.10:
Each module **SHALL** have less conducted emissions than the levels specified by the Max. Rated Dynamic Module Input Current in Figure B.45 on all power supply voltages used. For test method see B.8.7.3. I_{Dm} is the Dynamic Module Current of the module as specified by its manufacturer.

RULE B.8.11:
IF a module's I_{Dm} or I_{Pm} is dependent on a particular configuration or set of configurations,
THEN the module manufacturer **SHALL** document the configuration assumptions that allow the module to meet the conducted emissions specifications.

B.8.6.2 CONDUCTED SUSCEPTIBILITY

RULE B.8.12:
Each module **SHALL** operate as specified when subjected to conducted noise voltage levels specified in Figure B.46 on all power supply voltages used. For test method see B.8.7.4. V_{MI} is the Induced Ripple/Noise value specified in Table B.7. V_{ML} is the DC Load Ripple/Noise value specified in Table B.7.

RECOMMENDATION B.8.5:
Module Electrostatic Discharge (ESD) return path to chassis ground should be through the faceplate. The faceplate chassis ground is the primary return path, and energy to any bus pin should be at such a level as to not interfere with normal operation of mainframe or module e.g. change of state of data lines etc.

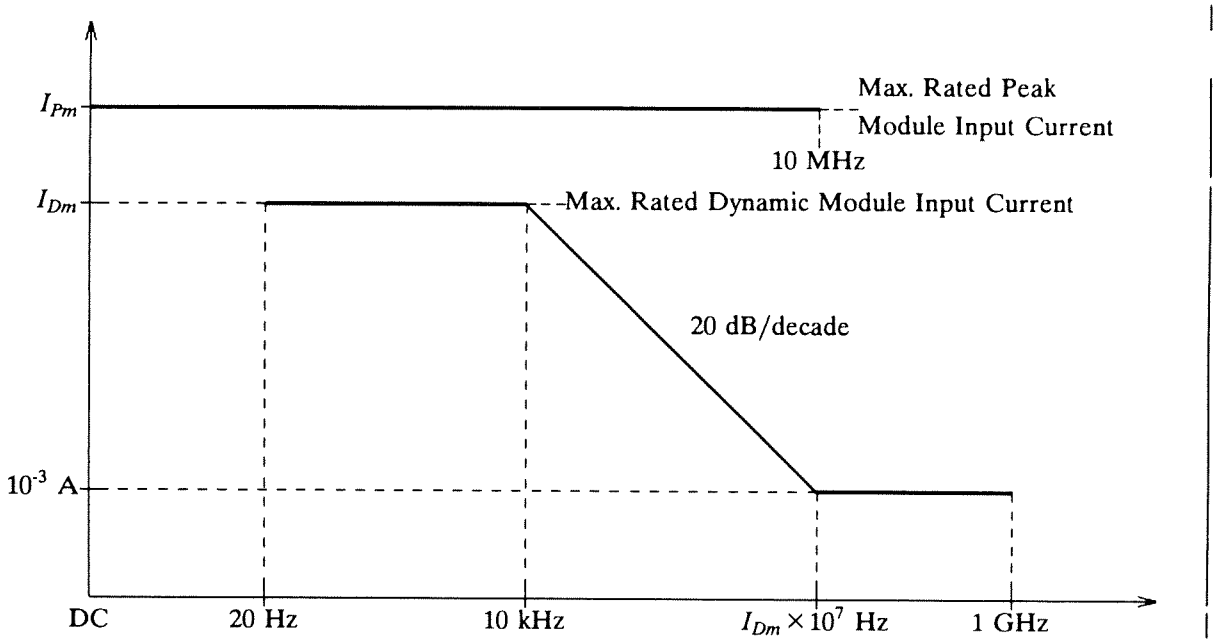


Figure B.45. Module Conducted Emissions

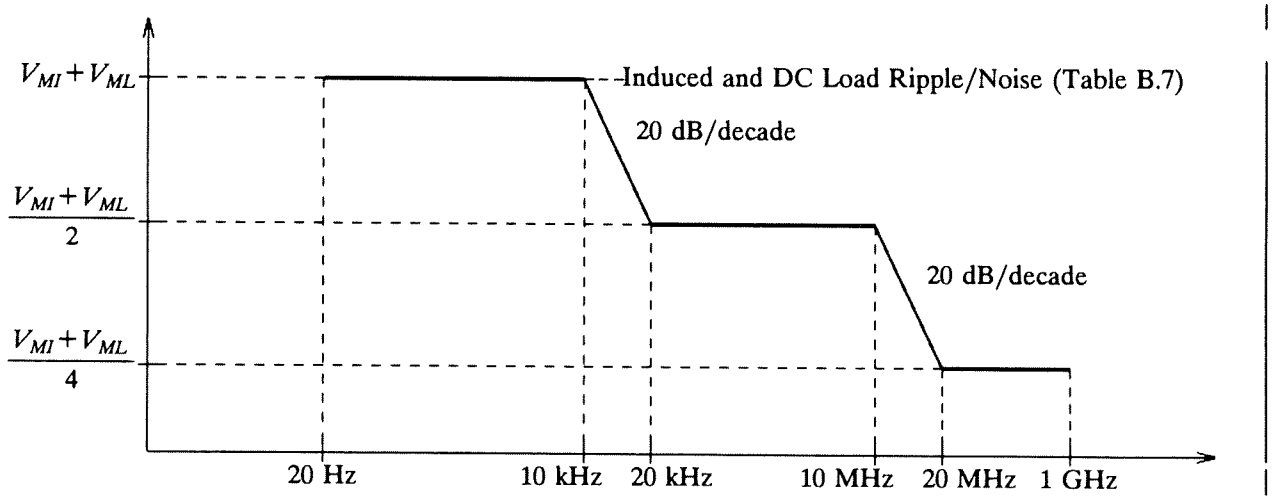


Figure B.46. Module Susceptibility Level

B.8.6.3 RADIATED EMISSIONS

Mainframe and modules will specify conformance to chosen standards appropriate to the intended market, e.g. FCC, VDE, or MIL Spec. for far field radiated emissions. Each module should not contribute more than its fair share of the total allowable radiation, i.e. in a 13 module mainframe, it should not contribute more than 1/13 of the allowable radiation if its radiation is coherent. Module to module interference is a close-field radiation problem. The term close-field is used to indicate those fields within a few centimeters of the surface of the modules.

RULE B.8.13:
The module manufacturer **SHALL** specify under what conditions the mainframe containing the module will meet the chosen radiated emission requirements.

RECOMMENDATION B.8.6:
Twelve identical single slot modules plus the Slot 0 module should stay within the chosen limits of the mainframe regulatory requirement. This is a coherency test.

OBSERVATION B.8.6:
If one module is tested alone the radiation per slot should be 1/13 of the allowable far field specification if the radiation is coherent. Module radiation allowed is proportional to the number of slots occupied.

OBSERVATION B.8.7:
If the module complies with the coherency test described in the above recommendation then the module manufacturer may just claim compliance to the regulatory radiation requirement with no additional documentation. If the module does not comply with the coherency test then the burden is on the module manufacturer to document under what configurations the mainframe containing the module will meet the regulatory radiation requirements.

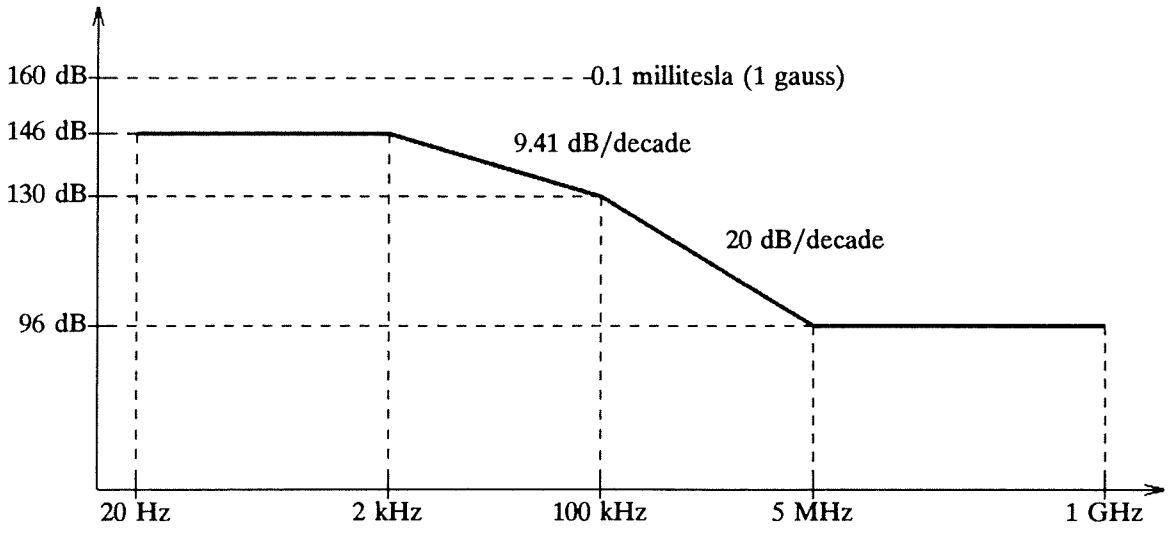


Figure B.47. A & B size Maximum close-field emissions (dB above 1 picotesla).

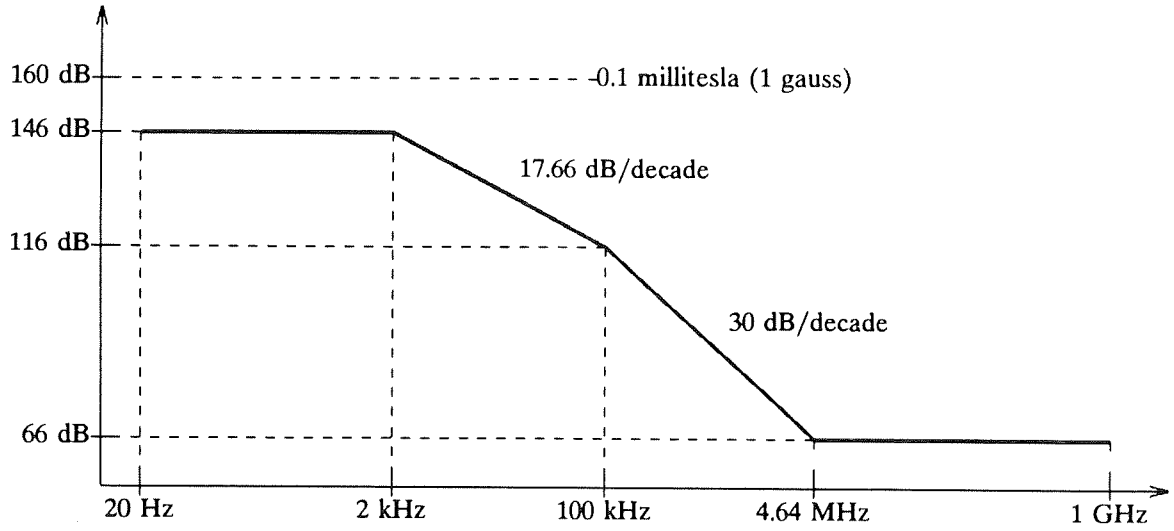


Figure B.48. C & D size Maximum close-field emissions (dB above 1 picotesla).

RULE B.8.14:
The close-field magnetic emissions of A size and B size modules when measured on the intermodule separation plane identified by the shaded area of Figure B.51 **SHALL NOT** exceed the limit as shown in Figure B.47. For test method see Section B.8.7.5, "Close-Field Magnetic Emissions Test".

RULE B.8.15:
The close-field magnetic emissions of C size and D size modules when measured on the intermodule separation plane identified by the shaded area of Figure B.52 **SHALL NOT** exceed the limit as shown in Figure B.48. For test method see Section B.8.7.5, "Close-Field Magnetic Emissions Test".

RULE B.8.16:
IF an A size or B size module is inserted in a C size or D size mainframe,
THEN the close-field magnetic emissions when measured on the C size or D size intermodule separation plane identified by the shaded area of Figure B.53 **SHALL NOT** exceed the limit as shown in Figure B.47. For test method see Section B.8.7.5, "Close-Field Magnetic Emissions Test".

RULE B.8.17:
IF an A size or B size module is inserted in a C size or D size mainframe,
THEN the close-field magnetic emissions when measured on the C size or D size intermodule separation plane identified by the shaded area of Figure B.54 **SHALL NOT** exceed the limit as shown in Figure B.48. For test method see Section B.8.7.5, "Close-Field Magnetic Emissions Test".

PERMISSION B.8.1:
A size and B size modules **MAY** use the shield of an adapter to meet the close-field magnetic emissions limit when installed into a C size or D size mainframe.

RECOMMENDATION B.8.7:
The A size and B size module manufacturer should document the available adapter(s) or procedure(s) that allow a module to meet the close-field magnetic compatibility limits when installed into a C size or D size mainframe.

OBSERVATION B.8.8:
The integration of A size VXI, B size VXI, or standard VME modules into C size or D size mainframes will require some forethought and planning. The system integrator should carefully consider adapter choice as well as module placement to preserve electromagnetic compatibility.

OBSERVATION B.8.9:
For modules without covers or shields, the surface for close-field testing is defined as the intermodule separation plane. The close-field magnetic emissions and susceptibility of the backplane is not specified.

RECOMMENDATION B.8.8:
Place all significant sources of radiated emissions, such as switching power converters or crystal oscillators, within an area 15 cm (6 inches) or less from the backplane connectors. Place all radiated emissions sources as close as possible to the backplane connectors and as far as possible from the board guides.

OBSERVATION B.8.10:
There is no close field EMC specification for backplanes. Module EMC requirements start 38 mm (1.5 inches) from the backplane surface. The connector shield described in the Mechanical Specification may be required to comply with the close field magnetic emission specification.

RECOMMENDATION B.8.9:
All C size and D size backplanes should provide a conductive surface in the grounding area described in the Mechanical Specification.

OBSERVATION B.8.11:
There is no specified limits for broadband or non-repetitive close-field emissions.

RECOMMENDATION B.8.10:
Place all significant sources of broadband emissions as close as possible to the backplane connectors, and as far as possible from the board guides.

B.8.6.4 RADIATED SUSCEPTIBILITY

Module to module interference is a close-field susceptibility problem. The term close-field is used to indicate those fields within a few centimeters of the surface of the modules. Close-field magnetic emissions and susceptibility levels should be met using good engineering design practices.

RULE B.8.18:
A size and B size modules **SHALL** operate as specified when subjected to magnetic fields of the level shown in Figure B.49, in the region of the intermodule separation plane identified by the shaded area of Figure B.51. For test method see Section B.8.7.6, "Close-Field Magnetic Susceptibility Test".

RULE B.8.19:
C size and D size modules **SHALL** operate as specified when subjected to magnetic fields of the level shown in Figure B.49, in the region of the intermodule separation plane identified by the shaded area of Figure B.53. For test method see Section B.8.7.6, "Close-Field Magnetic Susceptibility Test".

RULE B.8.20:
C size and D size modules **SHALL** operate as specified when subjected to magnetic fields of the level shown in Figure B.50, in the region of the intermodule separation plane identified by the shaded area of Figure B.54. For test method see Section B.8.7.6, "Close-Field Magnetic Susceptibility Test".

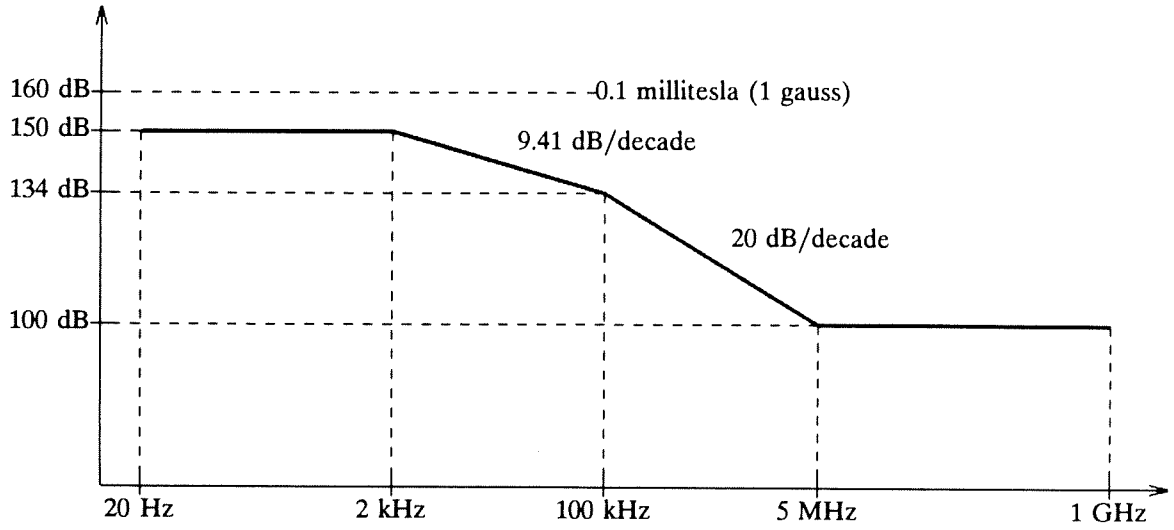


Figure B.49. A & B size Minimum close-field susceptibility (dB above 1 picotesla).

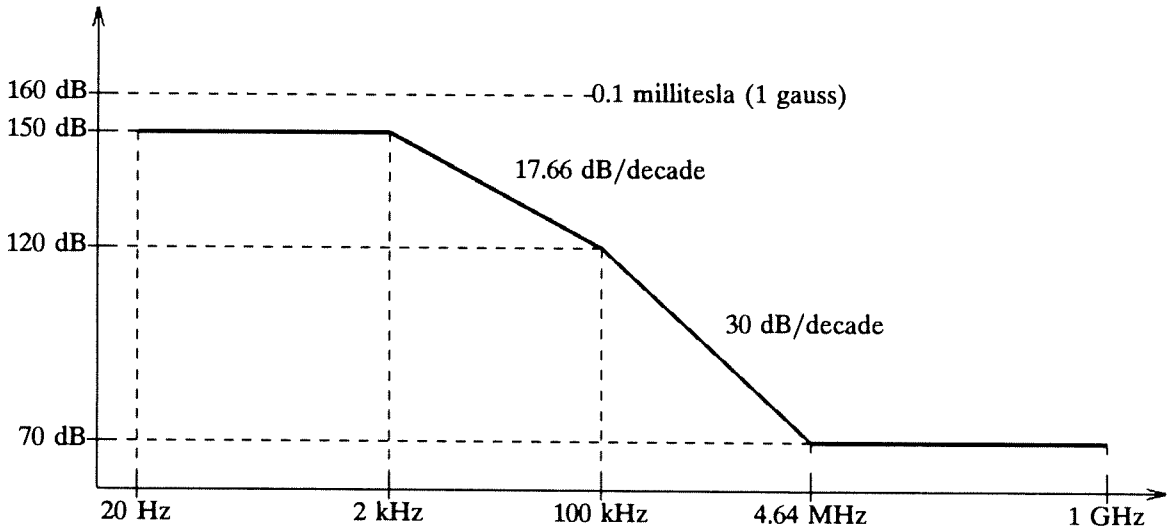


Figure B.50. C & D size Minimum close-field susceptibility (dB above 1 picotesla).

RECOMMENDATION B.8.11:

Place all circuits susceptible to radiated emissions, such as front end input circuits, in the area of a module greater than 15 cm (6 inches) from the backplane connectors. Place these circuits as close as possible to the front face plate and as far as possible from the board guides.

OBSERVATION B.8.12:

There are no specified limits for broadband or non-repetitive close field susceptibility.

RECOMMENDATION B.8.12:

Place all circuits susceptible to broadband or non-repetitive emissions as close as possible to the front face plate and as far as possible from the board guides.

RECOMMENDATION B.8.13:

Electric close-fields should be evaluated on all unshielded modules.

OBSERVATION B.8.13:

Electric field emissions are easily contained through shielding techniques i.e. Faraday shields. Unshielded modules should be checked for excessive electric field emissions, particularly at areas where high voltage swings are present. A length of coax with one or two inches of outer conductor stripped back at the tip is a useful close-field probe for making relative electric field measurements. A shield around the generating module, around the susceptible module, or between the modules will usually eliminate any electric field interference problems.

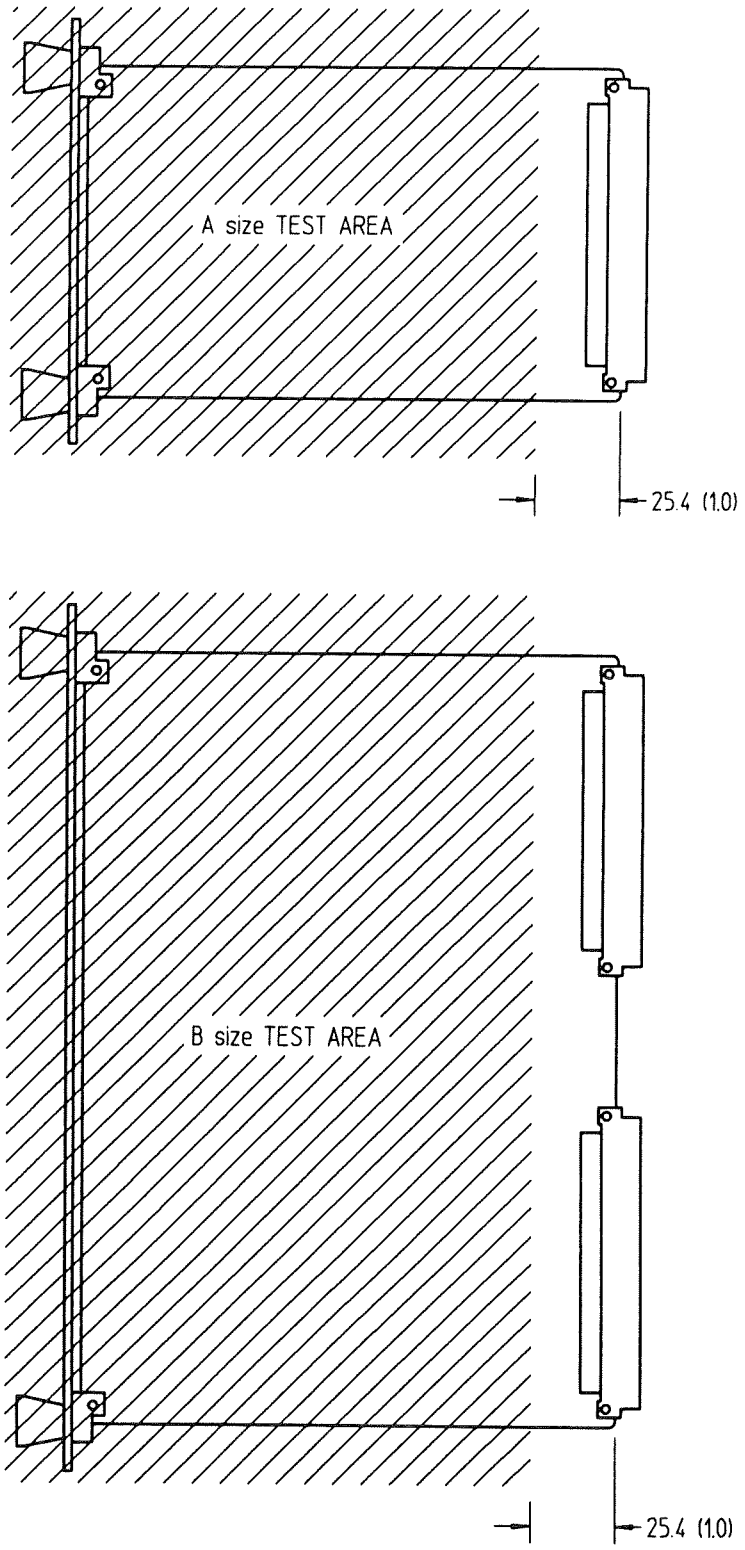


Figure B.51. A & B size EMC Test Areas.

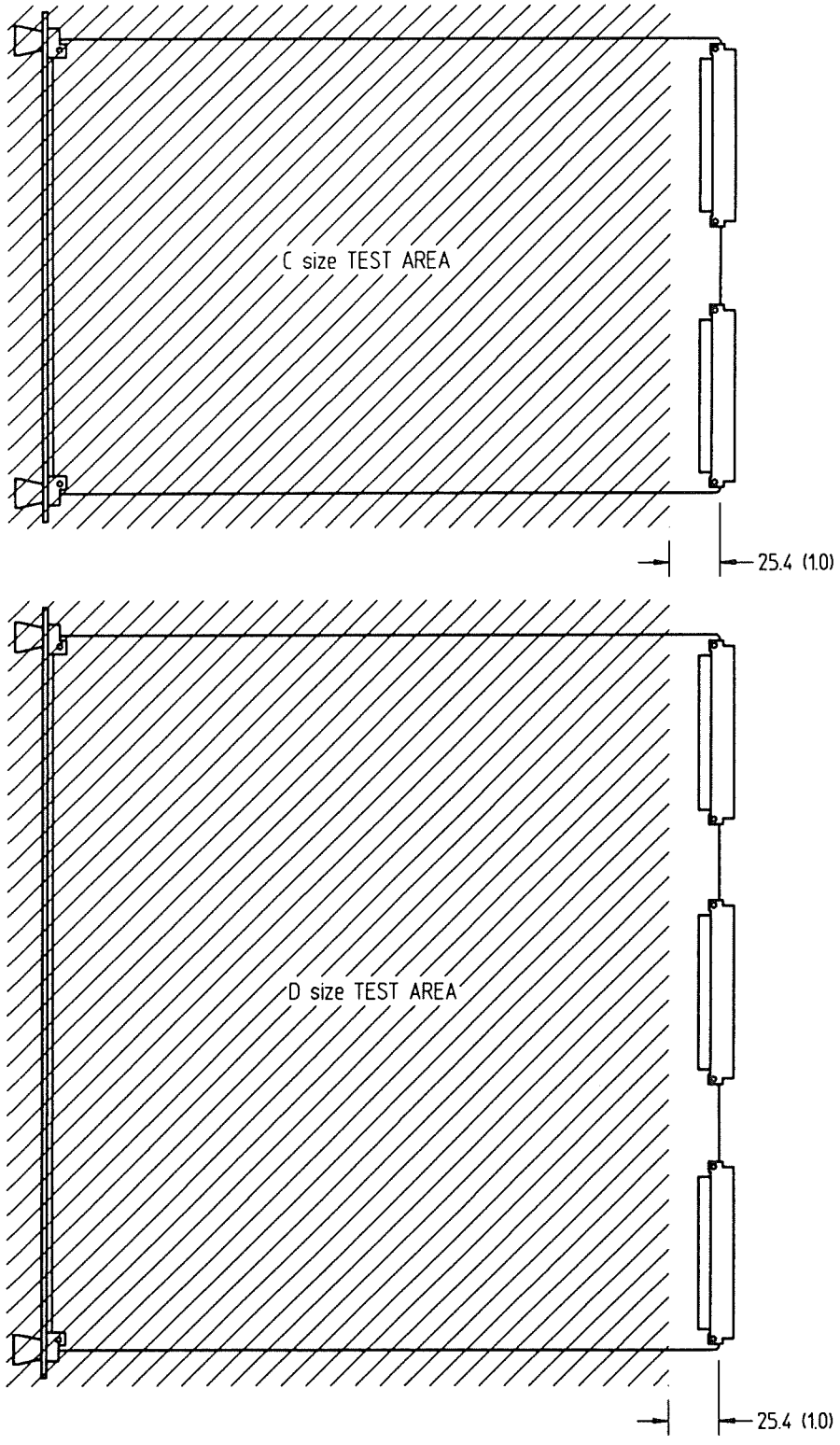


Figure B.52. C & D size EMC Test Areas.

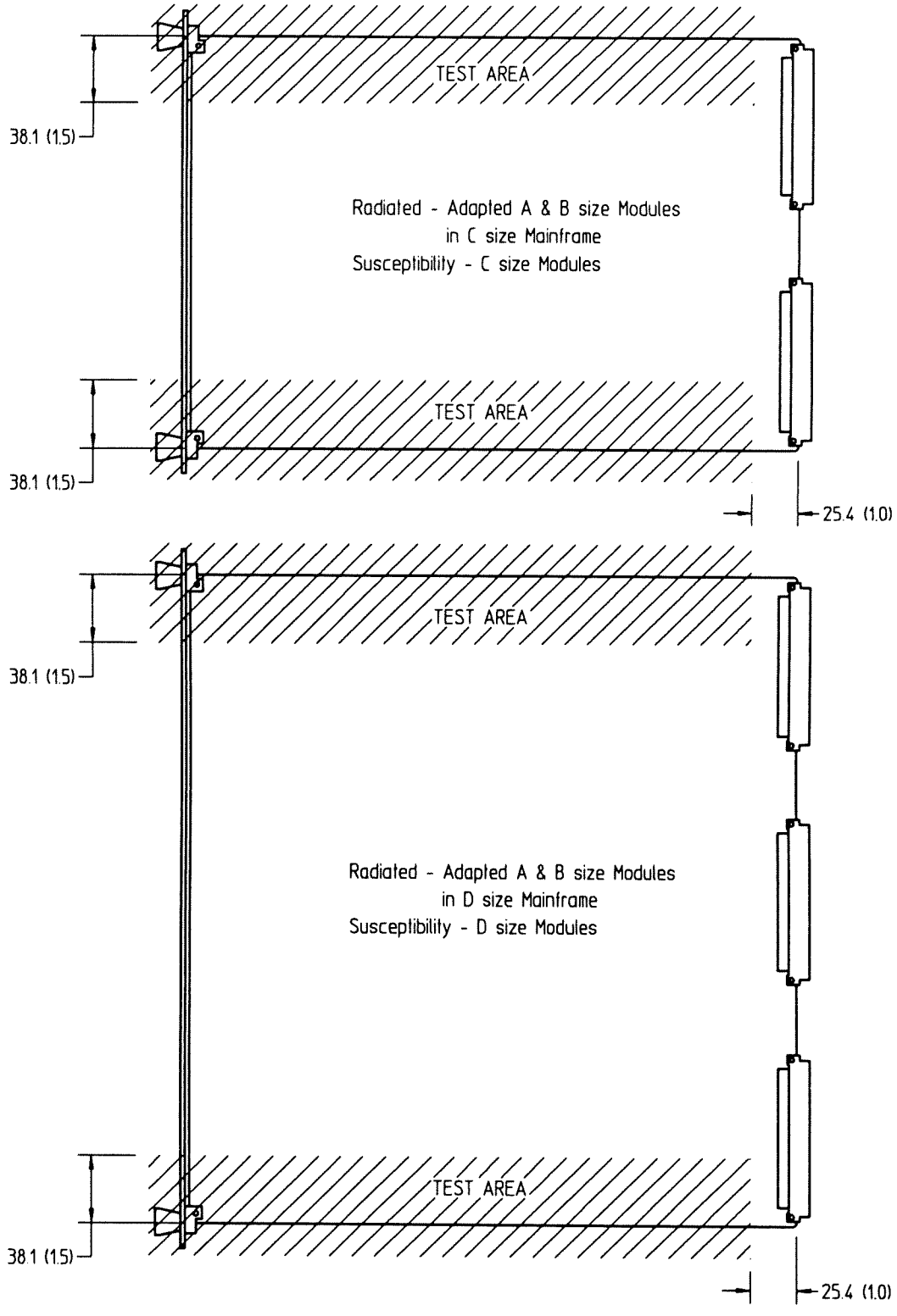


Figure B.53. Adapted A & B size & C & D size EMC Test Areas.

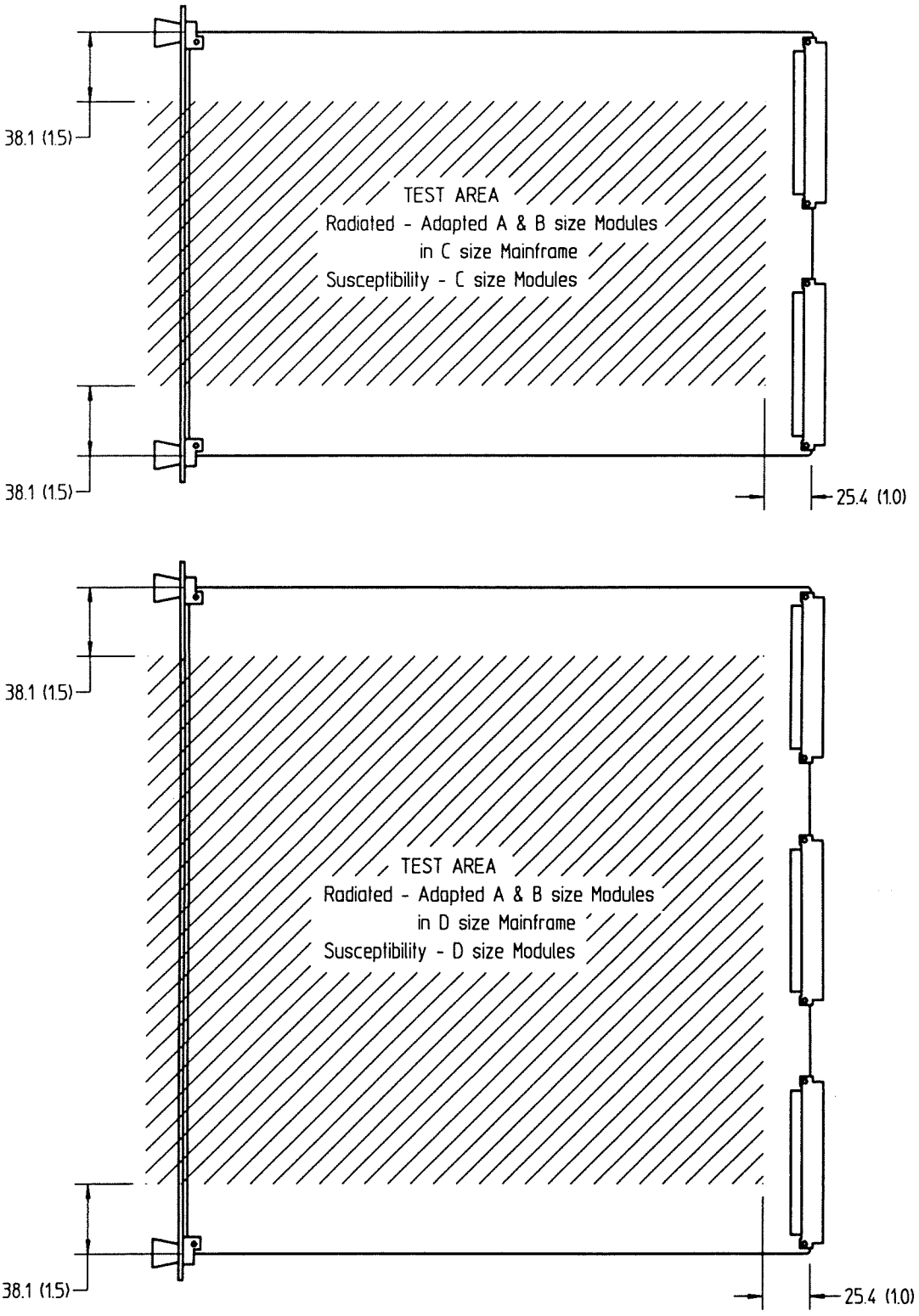


Figure B.54. Adapted A & B size & C & D size EMC Test Areas.

B.8.7 Suggested Test Methods

The test methods described in this section are provided as suggestions to the manufacturers of VXIbus mainframes and modules.

In the following sections, it is assumed that all measurements which are made using a spectrum analyzer, are for single-frequency stationary signals. Therefore the bandwidth and sweep rate must be chosen to accurately measure only one frequency component at a time. For modules or mainframes which may generate non-stationary signals, for example, swept signals or transients, the specified techniques cannot be used, but the manufacturer should point out in his documentation what limitations, if any, may arise because of these signals.

At higher frequencies, a relatively narrow bandwidth may be necessary to obtain sufficient sensitivity.

Note that all induced and conducted tests are based on peak-to-peak measurements, while the radiated tests are based on RMS measurements.

B.8.7.1 DC LOAD RIPPLE/NOISE TEST OF MAINFRAMES

The DC load ripple/noise is the maximum periodic and random devictions (PARD) in a power supply voltage under a constant DC load, as used within the power supply industry. Using an oscilloscope with a 10 Mhz bandwidth, measure the peak-to-peak AC voltage on each power supply output while varying the DC load from 0 to the mainframe peak current (I_{MP}). The highest value measured is the DC load ripple/noise. The probe should be connected to the backplane using good RF techniques, i.e. using very short leads.

B.8.7.2 INDUCED RIPPLE/NOISE TEST OF MAINFRAMES

The Induced Ripple/Noise test requires applying a load of the magnitude and frequencies shown by the Max. Rated Mainframe Dynamic Output Current in Figure B.43 while monitoring the power supply voltage to guarantee that the total induced and DC load ripple voltage does not exceed that of Figure B.44.

Set the DC load on the power supply to be half of the rated current (½ I_{MP}). For the low frequency tests (up to 10 kHz), drive an active load with a low frequency oscillator to the levels specified. Measure the noise voltage present at an adjacent slot using an oscilloscope. The total induced and DC load ripple/noise is the peak-to-peak value of the AC component. For the higher frequency conducted emission tests, drive a load from a tracking oscillator while monitoring the induced voltage at an adjacent slot with a spectrum analyzer.

B.8.7.3 CONDUCTED EMISSIONS TEST OF MODULES

The testing of a module's dynamic current should exclude all effects of the mainframe. Because the mainframe provides current for backplane terminations, clocks, and other devices requiring dynamic current, voltage ripple may exist on power pins at various frequencies. This power supply ripple voltage may cause an apparent dynamic current flow into the module that is not caused by any current requirements of the module, and should not be included as part of the module's dynamic current. By supplying voltages to the module under test from independent power supplies, the mainframe ripple performance will be isolated from the module dynamic current requirements.

The module being tested should be operating at its worst case current versus frequency requirements.

A current sensor in series with each supply of the module is used to measure the AC portion of the power supply load. Check with an oscilloscope that the module meets the low frequency (up to 10 KHz) conducted emissions requirements. A Spectrum Analyzer is used to measure the higher frequency content of the current. The module is exercised to find the highest readings. *

This test should be conducted using a backplane having an impedance of less than $0.2 \times V_S/I_S$ in parallel with a ≥ 0.1 μF bypass capacitor at the measured slot, where V_S is the Module Susceptibility voltage level (Figure B.46) and I_S is the Module Conducted current level (Figure B.45).

B.8.7.4 CONDUCTED SUSCEPTIBILITY TEST OF MODULES

Induce a voltage between each backplane power supply and module at the maximum voltage shown in Figure B.44 as measured from the module supply pin to ground. Search for potential frequencies where performance is degraded due to the presence of injected noise. For low frequency testing, use a series resistor driven by an active load to induce the specified voltage level. For higher frequencies use current probes appropriate to the frequency range being tested. Be sure the current probe is rated for the DC current being supplied to the module.

B.8.7.5 CLOSE-FIELD MAGNETIC EMISSIONS TEST

The equipment to measure the close-field magnetic emissions of a module is a close-field magnetic probe and a spectrum analyzer that operates over the frequency range specified in Figure B.47 or Figure B.48. To cover the frequency range, different probes and spectrum analyzers may be required. It is required that the close-field probes have calibrated antenna factors and that the magnetic loop be within 5 mm of the end of the probe. With the close-field magnetic probe connected to the spectrum analyzer, move the probe tip over the entire module surface to search for potential radiating spots such as seams and holes. At all areas where magnetic radiation is detected, the probe should be oriented to maximize the probe output as displayed on the spectrum analyzer, Using the antenna factors of the close-field probe, calculate the magnetic flux density for each frequency of radiation detected. Flux density levels may not exceed the levels of the appropriate figure.

B.8.7.6 CLOSE-FIELD MAGNETIC SUSCEPTIBILITY TEST

The equipment to test the susceptibility of a module consists of a close-field magnetic probe connected to a signal generator that operates over the frequency range specified in Figure B.49 or Figure B.50. To cover the frequency range, different probes and signal generators may be required. It is required that the close-field probes have calibrated antenna factors and that the magnetic loop be within 5 mm of the end of the probe. Connect the close-field magnetic probe to the signal generator. For each frequency tested, set the signal generator to produce a magnetic flux density at the limit given by the appropriate figure for that frequency. Move the probe tip over the entire module surface to search for potential spots where the performance of the module is degraded due to the presence of a magnetic field. The probe tip should be within 1 cm of the surface of the module.

B.8.7.7 EXAMPLES OF CLOSE FIELD MAGNETIC PROBES

Examples of close-field probes that meet the above requirements are:

Magnetek Part Number 1846 for frequencies from 20 Hz to 10 kHz

Hewlett-Packard Model 11941 for frequencies from 10 kHz to 30 MHz

Hewlett-Packard Model 11940 for frequencies from 30 MHz to 1 GHz

EMCO Model 7405 Near-Field Probe Set may also be used to complement the above probes.

An amplifier may be needed with any probe to compensate for large probe antenna factors.

C. SYSTEM ARCHITECTURE:

C.1 VXIbus SYSTEM ARCHITECTURE OVERVIEW

The VXIbus architecture allows a wide range of instruments, interface cards, or computers from different manufacturers to coexist as modules in the same cardcage. The range of the applications being addressed by different cardcages in the VXIbus family is very broad and hence the architecture is intentionally left open to accommodate this flexibility. VXIbus does not define a specific system hierarchy or topology nor does it specify the type of microprocessor, operating system or the type of interface to the host computer. What VXIbus defines is a foundation (or platform) upon which the above decisions can be made while ensuring compatibility between different manufacturers. Figure C.1. shows *some of the* possible system configurations and may help illustrate the flexibility that is allowed by the VXIbus standard.

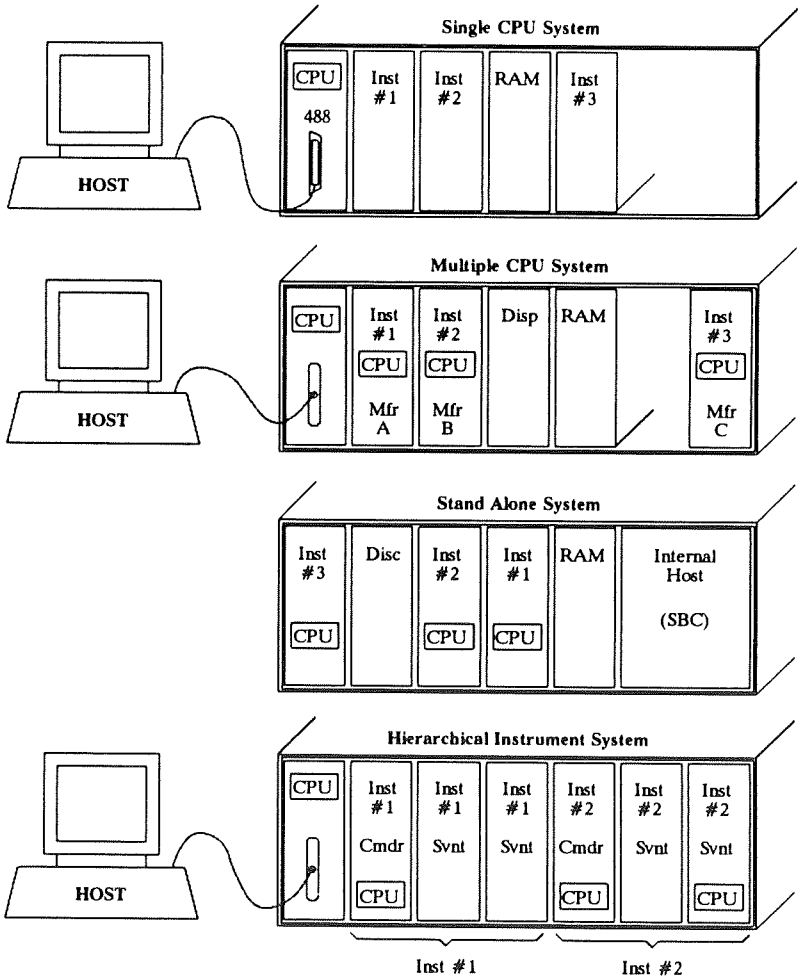


Figure C.1. Typical System Configurations.

As shown by the examples in Figure C.1. there are many topologies possible within the VXIbus architecture. These include single CPU systems that centralize the intelligence for all their instrument modules, multiple CPU systems with distributed intelligence, and stand alone mainframes that contain a host computer running user applications. Each of these topologies has different communication needs which are met by a layered set of communication protocols illustrated in Figure C.2.

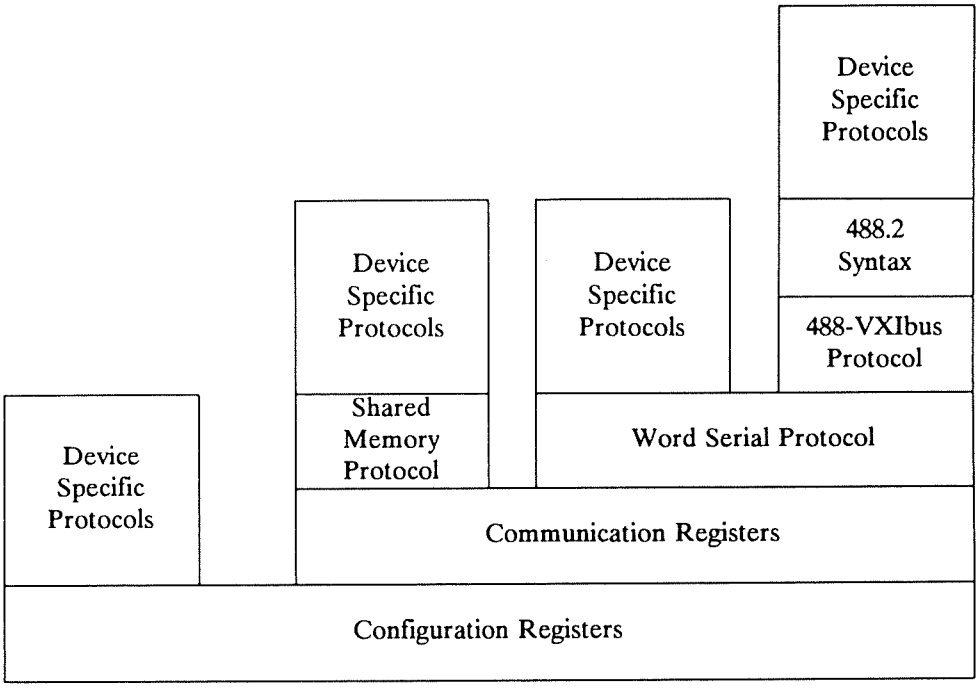


Figure C.2. VXIbus Communication Layers

In order to support automatic system and memory configuration, the VXIbus standard specifies a minimum amount of capability on each device regardless of its function. A VXIbus device (see Section C.2, "Device Operation"), being the lowest element of a hierarchy has a set of "Configuration Registers", all totally accessible from P1 on VMEbus which allow the system to identify the device, its class, model and manufacturer, address space (A16, A24, A32), and memory requirements. VXIbus devices with only this minimum level of capability are called **REGISTER BASED DEVICES.** As an example, the single CPU system shown in Figure C.1. may implement all of its instruments using only Register Based Devices with device specific protocols on how the CPU communicates with each of those instruments. VXIbus also defines **MEMORY DEVICES**, (See Section C.2.3, "Memory Devices") that can be identified as RAM, ROM, or other memory types and be configured in contiguous blocks of memory based on speed or memory type.

If a higher level of communication ability between modules is desired in the system, VXIbus defines a class of devices called **MESSAGE BASED DEVICES**. In addition to the configuration registers mentioned above, Message Based Devices are required to have communication registers that are accessible to other modules in the system. Each device in the system can then use specific communication protocols such as VXIbus's Word Serial Protocol (See Section C.3.3.1, "Word Serial Protocols") to communicate with other devices. The multi-CPU system shown in Figure C.1. is a good example of this type of system where each instrument is a Message Based Device and is therefore capable of receiving instructions from a host or a common host interface. Note that by agreeing on a class of communication protocols, such as Word Serial, different manufacturers can now build any of those instruments and be assured of compatibility with the rest of the system. Higher level communication protocols referred to as Shared Memory Protocols in Figure C.2. can later be defined using these platforms.

Communication between VXIbus devices is based on a hierarchical device relationship involving **COMMANDERS** and **SERVANTS**. For many systems such as the single CPU system shown in Figure C.1. only the first level of this hierarchy exists. In that example, the CPU and host interface device is a Commander which controls the 3 Instrument Servants. A Commander is able to initiate communication with its Servants based on their capabilities. If the Servants are Message Based Devices, such communication can take place using **COMMANDS** which are described by the Word Serial Protocols. Communication with Register Based Servants is device dependent and may vary between systems. The Commander/Servant hierarchy can be nested since a Message Based Device may be a Commander as well as a Servant to the next level above it. The hierarchical instrument system of Figure C.1. is an example of such a system. Instrument #1 and #2 are shown as multi device instruments that have two Servants each while the instruments themselves may be Servants to the host interface.

A common element that instruments may desire to share among them is the interface to the host computer as shown in the Multi-CPU system example in Figure C.1. The 488-VXIbus interface device (see Section D.2, "488-VXIbus Interface") is an example of such a device. VXIbus defines this as a function specific Message Based Device with communication protocols that support IEEE-488 functions. This device, if provided, may be shared by the instruments needing to be accessed by the host. Other host or peripheral interfaces such as RS-232, LAN, or Human Interfaces can be designed in a similar fashion.

Regardless of the topology of the VXIbus cardcage, (See Figure C.1.) VMEbus's multiple bus master architecture demands a minimal amount of system related functions that are performed by a central RESOURCE MANAGER (see Section C.4.1, "Resource Manager") The following are some of the functions a Resource Manager MAY need to perform depending on the topology of the system.

- *Address map configuration.* To configure the A24, A32 address maps and assign blocks of addresses to the devices requiring them.
- *Determining System Hierarchy.* In a multi-CPU system, the potential for more than one processor to try and control another device exists. The Commander/Servant hierarchies determined by the Resource Manager prevent this type of conflict.
- *Allocating Shared System Resources* . A central Resource Manager (e.g. operating system) is needed in systems that share blocks of memory for communication or share other hardware resources such as trigger busses.
- *Performing System Self Test and Diagnostics.*
- *Initialization of all Commanders in the system.*

C.2 DEVICE OPERATION

C.2.1 Device Overview

Devices are the lowest logical component in the VXIbus system. Normally a device will consist of one VXIbus module. However, multi-board devices and multi-device boards are permitted. There is a unique Logical Address associated with each device in the system. Examples of devices are computers, multimeters, multiplexers, oscillators, operator interfaces, and counters. VXIbus devices may be categorized by their supported protocols into four classes as illustrated in Figure C.3.

- Message Based Devices support the VXIbus configuration and communication protocols. This category includes only devices with Commander and/or command based Servant elements. Examples of Message Based Devices are any devices with local intelligence that require a level of communication capability such as DMM's, Spectrum Analyzers, Display Controllers, 488-VXIbus interface devices, Switch controllers, etc.
- Register Based Devices support VXIbus register maps, but no VXIbus communication protocols. This category is comprised of devices having Register Based Servant elements. Typically Register Based Devices are simple, inexpensive devices such as simple switches, digital I/O cards, simple serial interface cards and in general any card that requires little or no local intelligence. More sophisticated devices could also be implemented as Register Based Devices that rely on Message Based Devices for their instructions.
- Memory Devices have configuration registers and contain certain attributes of a Memory Device such as type and access time, etc. but have no other VXIbus defined registers or protocols. Bubble memory, RAM and ROM cards are in this category.
- Extended Devices are special purpose VXIbus devices that have configuration registers so they can be identified by the system. This category of devices will allow for definition of future device classes to support further levels of compatibility.

Hybrid Devices are VMEbus compatible devices that know about VXIbus devices and have the ability to communicate with them or make use of them, but don't themselves comply with VXIbus device requirements. Existing VMEbus boards with the appropriate software to make use of VXIbus devices are an example of this class of device.

Non-VXIbus devices are VMEbus devices that do not comply with or use any of the VXIbus requirements.

C.2.1.1 DEVICE SLAVE CAPABILITIES

All VXIbus devices have a common set of VMEbus slave attributes. These include standard memory maps, common addressing modes, and standard register definitions.

C.2.1.1.1 Device Addressing

All VXIbus devices have registers located within 64 byte blocks in the A16 address space. The base address of a device's registers is determined by the device's unique Logical Address. The Logical Address selector is an eight (8) bit selector which has 256 unique settings. The Logical Address is the value indicated by this selector, and corresponds to bits 6 → 13 of the device register base address. Bits 14 and 15 of the base address are both 1, and the base address is

$$V*64+49152$$

where V is the device's Logical Address.

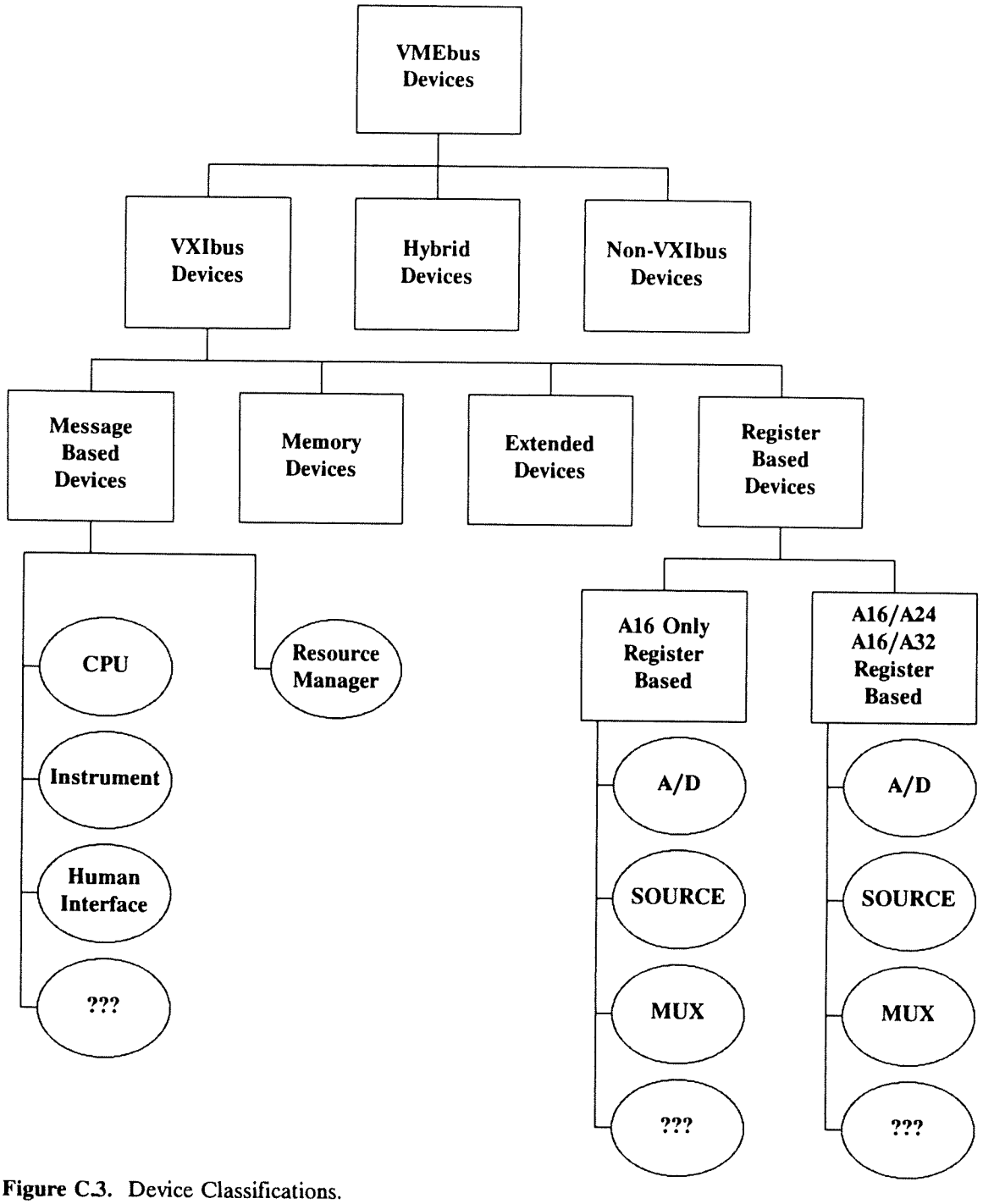


Figure C.3. Device Classifications.

If a device requires fewer than 64 bytes of address space, all of it may reside within the defined 64 byte block. Otherwise, it has additional registers in the A24 or A32 address space. The base address of a device's A24 or A32 registers is programmed via an Offset register in the configuration portion of its A16 registers.

RULE C.2.1:
Every VXIbus device **SHALL** have a non-volatile means of selecting Logical Addresses and implement the defined A16 addressing scheme.

RULE C.2.2:
IF a VXIbus device has A24 or A32 registers,
THEN the base address of such registers **SHALL** be programmable via its Offset register in its A16 configuration area.

RULE C.2.3:
All VXIbus slaves **SHALL** implement A16 addressing modes and have configuration registers available in the A16 address space.

PERMISSION C.2.1:
VXIbus slaves **MAY** also implement A24 or A32 addressing modes.

RULE C.2.4:
IF a VXIbus slave implements A24 addressing modes
THEN it **SHALL NOT** implement A32 addressing modes.

OBSERVATION C.2.1:
IF a VXIbus slave implements A32 addressing modes
THEN it cannot implement A24 addressing modes.

RULE C.2.5:
VXIbus slaves **SHALL** implement D16.

RECOMMENDATION C.2.1:
VXIbus slaves should implement D08 (EO).

PERMISSION C.2.2:
VXIbus slaves **MAY** implement D32.

IF	THEN					
	A16	A24	A32	D08(EO)	D16	D32
SLAVE A16	●	MAY	MAY	REC.	SHALL	MAY
SLAVE A24	SHALL	●	N.A	REC.	SHALL	MAY
SLAVE A32	SHALL	N.A.	●	REC.	SHALL	MAY

N.A.- Not Allowed
REC. - Recommended

C.2.1.1.2 Configuration Registers

The following diagram contains the register map for the Configuration registers of a VXIbus device. These registers are located in the A16 address space by the Logical Address selector as described previously.

$3F_{16}$	DEVICE DEPENDENT REGISTERS
$1F_{16}$	DEVICE CLASS DEPENDENT REGISTERS
06_{16}	Offset Register
04_{16}	Status/Control Register
02_{16}	Device Type
00_{16}	ID/Logical Address Register

The fields of the above table are defined as follows:

ID Register: A read of this 16 bit register provides information about the device's configuration. Its contents are defined in the following table.

Bit #	15 ← 14	13 ← 12	11 ← 0
Contents	Device Class	Address Space	Manufacturer ID

- Device Class: This field indicates the classification of the VXIbus device according to the following table.

Value	**Class**
00	Memory
01	Extended
10	Message Based
11	Register Based

- Address Space: This field indicates the addressing mode(s) of the device's operational registers according to the following table.

Value	**Mode**
00	A16/A24
01	A16/A32
10	RESERVED
11	A16 Only

- Manufacturer ID: This number uniquely identifies the manufacturer of the device. The list of ID numbers is maintained by the VXIbus Consortium. Each VXIbus device manufacturer has exactly one Manufacturer ID number. Numbers are assigned to manufacturers in decreasing order beginning with number 4095. See Section A.1, "Manufacturer ID Numbers", for information on obtaining a manufacturer ID number.

OBSERVATION C.2.2:

ID number 3840 ($F00_{16}$) is reserved for user customized devices and any other unique special function device.

Logical Address: This 16 bit write register is defined by the optional Dynamic Configuration protocol. See Section F, "Dynamic Configuration".

Device Type: This 16 bit register contains a device dependent type identifier. The read contents of this register are defined below

Bit #	15 ← 12	11 ← 0
Contents	Required Memory	Model Code

- Required memory [Only required for A16/A24 and A16/A32 devices] : These 4 bits contain a number *m*, which is between 0 and 15. The required memory usage is defined as

$$256^{a} * 2^{23-m},$$

where *a* is the value of the *Address Space* field in the ID register. This equation gives the amount of A24 or A32 memory space (in bytes)resident on the device. This also means that the device must decode addresses to this resolution.

OBSERVATION C.2.3:

The above algorithm for determining memory requirements allows devices with minimal memory requirements to let the *Required Memory* bits float to a high level and minimizes hardware requirements for low cost devices.

In the case of an A16 only device, these four bits are the upper bits of the Model Code.

OBSERVATION C.2.4:

The minimum size of an A24 device's memory is 256 bytes. The maximum size of the memory map is ½ of the complete A24 VMEbus address space.

OBSERVATION C.2.5:

The minimum size of an A32 device's memory is 65536 bytes (64 Kbytes). The maximum size of the memory map is ½ of the complete A32 VMEbus address space.

SUGGESTION C.2.1:

It would be a good idea to limit the memory space of any device to ¼ of the available address range for the device. This would facilitate having more than one device in each possible address space.

- Model Code: This field contains a unique card identifier which is defined by the manufacturer. In the case of an A16 only device, this field occupies all 16 bits of the Device Type register.

OBSERVATION C.2.6:

Model codes 0-255 ($0\text{-}FF_{16}$) are reserved for Slot 0 devices. See Rules C.4.16 and C.4.17 in Section C.4.3, "VXIbus Subsystem Slot 0".

A write to the Device Type register location has no effect. Its use is reserved for VXIbus definition.

Status Register: A read of this register provides information about the Device's status as defined in the following table.

Bit #	15	14	13 - 4	3	2	1 - 0
Contents	A24/A32 Active	MODID*	Device Dependent	Ready	Passed	Device Dependent

- A24/A32 Active: This bit is only required for A16/A24 or A16/A32 devices. A one (1) in this field indicates that its A24 or A32 registers can be accessed. This bit reflects the state of the Control register's *A24/A32 Enable* bit.

RULE C.2.6:
The *A24/A32 Active* bit in an A16/A24 and A16/A32 device **SHALL** be cleared (set to zero (0)) whenever SYSRESET* is asserted, and changed only by a write of the *A24/A32 Enable* bit in the device's Control register.

OBSERVATION C.2.7:
In an A16 only device the *A24/A32 Active* bit is device dependent.

- MODID*: A one (1) in this field indicates that the device is not selected via the P2 MODID line. A zero (0) indicates that the device is selected by a high state on the P2 MODID line.

RULE C.2.7:
IF A device does not implement MODID capability (P1 only device)
THEN the device's *MODID** bit **SHALL** always be set to one (1).

RULE C.2.8:
IF A device implements MODID capability,
THEN the device's *MODID** bit **SHALL** always be set or reset within 1 μs of the module's MODID line being set or reset.

- Ready: This field serves two purposes:

1. After the power-on initialization sequence (see Section C.2.1.2, "Device Initialization and Diagnostics") a one (1) in this field in combination with a zero (0) in the *Passed* field indicates that the device has failed register initialization.
2. After the power-on initialization sequence, a one (1) in this field in combination with a one (1) in the *Passed* field indicates that the device is ready to execute its full set of operational commands.

OBSERVATION C.2.8:
The *Ready* bit indicates that a device is ready to accept its full set of operational commands. This "readiness" is determined in a device dependent manner. A standard definition of readiness for Message Based Devices is given in Section C.2.4.5, "Message Based Device Configuration".

- Passed: A zero (0) in this field indicates that the device is either executing or has failed it's self test. A one (1) indicates that the self test has successfully completed.

RULE C.2.9:
IF a device does not implement a power-on self test,
THEN its *Passed* bit **SHALL** always be set to one (1).

PERMISSION C.2.3:
The device dependent bits of the Status register **MAY** be defined to indicate the state of the corresponding bits of the Control register.

Control Register: A write to this register causes specific actions to be executed by the device.

These actions are described in the following table.

Bit #	15	14 ← 2	1	0
Contents	A24/A32 Enable	Device Dependent	Sysfail Inhibit	Reset

- A24/A32 Enable: A one (1) in this field enables access to the device's A24 or A32 VMEbus registers. A zero(0) disables such access.
- Device Dependent: The bits in this field are available to the device designer for device and/or application specific control.

OBSERVATION C.2.9:
A Resource Manager which lacks device specific knowledge of a device will always write a one to each of the device dependent bits of a device's Control register whenever it writes to that Control register to change any of the VXIbus defined bits.

RECOMMENDATION C.2.2:
The device dependent bits of the Control register should be implemented so that writing ones to these bits will not disturb the initial state of the device. This will permit a Resource Manager or Commander (which lacks device specific knowledge) to write the VXIbus defined bits without causing some undesired activity on the device.

- Sysfail Inhibit: A one (1) in this field disables the device from driving the SYSFAIL* line.
- Reset: A one (1) in this field forces the device into a reset state.

RULE C.2.10:
IF a device's Commander writes a one (1) to the device's *Reset* bit,
THEN the Commander must not write a zero (0) to that device's *Reset* bit within the following 100 μs.

See Section C.2.1.2, "Device Initialization and Diagnostics", for more information on the operation of the above bits.

Offset Register: This register is needed only for A16/A24 and A16/A32 devices. This 16 bit read/write register defines the base address of this device's A24 or A32 operational registers. The $m+1$ most significant bits of the Offset register are the values of the $m+1$ most significant bits of the device's A24 or A32 register addresses, where m is the value of the *Required Memory* field of the device's Device Type register. The 15-m least significant bits of the Offset register are meaningless. Thus, the Offset register bits 15 → 15-m map to the address lines $A_{23} \rightarrow A_{23\text{-}m}$ for A24 registers, or to lines $A_{31} \rightarrow A_{31\text{-}m}$ for A32 registers.

OBSERVATION C.2.10:
In the case of an A16 only device, the Offset register's location is a device dependent operational register.

OBSERVATION C.2.11:
A read of the Offset register always returns the address offset most recently written to the Offset register.

OBSERVATION C.2.12:
The Dynamic Configuration protocol defines additional use of the Offset register. See Section F, "Dynamic Configuration".

C.2.1.1.3 Device Class Dependent Registers

The definition of the VXIbus Device Class Dependent Registers varies with the class of the device, i.e Message Based, Register Based, etc..

C.2.1.1.4 Device Dependent Registers

VXIbus Device Dependent Registers are user definable.

C.2.1.1.5 Address Modifiers

RULE C.2.11:
VXIbus device A16 registers **SHALL** respond to address modifiers 29_{16} (Short Non-Privileged Access) and $2D_{16}$ (Short Supervisory Access) only.

RULE C.2.12:
VXIbus device A24 registers **SHALL** respond to address modifiers $3D_{16}$ (Standard Supervisory Data Access) and $3E_{16}$ (Standard Supervisory Program Access).

RECOMMENDATION C.2.3:
VXIbus device A24 registers should also respond to address modifiers 39_{16} (Standard Non-Privileged Data Access) and $3A_{16}$ (Standard Non-Privileged Program Access).

PERMISSION C.2.4:
VXIbus device A24 registers **MAY** respond to either of the address modifiers $3B_{16}$ (Standard Non-Privileged Block Transfer) and $3F_{16}$ (Standard Supervisory Block Transfer).

RULE C.2.13:
VXIbus device A24 registers **SHALL NOT** respond to any address modifier other than 39_{16}, $3A_{16}$, $3B_{16}$, $3D_{16}$, $3E_{16}$, or $3F_{16}$.

RULE C.2.14:
VXIbus device A32 registers **SHALL** respond to address modifiers $0D_{16}$ (Extended Supervisory Data Access) and $0E_{16}$ (Extended Supervisory Program Access).

RECOMMENDATION C.2.4:
VXIbus device A32 registers should also respond to address modifiers 09_{16} (Extended Non-Privileged Data Access) and $0A_{16}$ (Extended Non-Privileged Program Access).

PERMISSION C.2.5:
VXIbus device A32 registers **MAY** respond to either of the address modifiers $0B_{16}$ (Extended Non-Privileged Block Transfer) and $0F_{16}$ (Extended Supervisory Block Transfer).

RULE C.2.15:
VXIbus device A32 registers **SHALL NOT** respond to any address modifier other than 09_{16}, $0A_{16}$, $0B_{16}$, $0D_{16}$, $0E_{16}$, or $0F_{16}$.

OBSERVATION C.2.13:
The preceding rules prevent any overlapping of the defined VMEbus address spaces. They do not prohibit the use of VMEbus user defined address modifiers within a VXIbus system.

C.2.1.2 DEVICE INITIALIZATION and DIAGNOSTICS

The VMEbus provides a minimal self-test reporting facility (SYSFAIL*). VXIbus devices implement an extended reporting strategy that addresses the ambiguities of the VMEbus specification. In addition, the VXIbus initialization process provides for a VXIbus device to initialize its own configuration registers.

C.2.1.2.1 Overview

The VXIbus initialization sequence provides the following services:

- Assertion of SYSFAIL* at power-on to indicate a self-test condition.
- Status register indication of self-test status.
- Optional faceplate LED indication of self-test status, labeled Failed.
- Time limit for completion of a normal self-test.
- Control register bits for disabling SYSFAIL* driver and resetting the device.
- A waiting period for initialization of the device's configuration registers.

The initialization sequence is implemented with an internal state machine, the *Passed* and *Ready* bits of the Status register, the *Reset* and *Sysfail Inhibit* bits of the Control register, a VMEbus SYSFAIL* driver, and the Failed LED.

A typical power-on initialization sequence proceeds as follows.

1. At power-on SYSRESET* is asserted, forcing the device into a reset condition. At this time the *Passed*, *Reset*, *Ready*, and *Sysfail Inhibit* bits are all cleared (to 0). The device asserts SYSFAIL* and lights its Failed LED (if present).
2. When SYSRESET* is de-asserted, the device begins its self test. This self test initializes the device's capability to execute VXIbus communication.
3. If the device fails its self test, it leaves its *Passed* and *Ready* bits cleared, SYSFAIL* asserted, and its Failed LED lighted. After 4.9 seconds, these conditions indicate a self test failure.
4. When the device successfully completes its self test, it sets (to 1) its *Passed* bit, de-asserts SYSFAIL*, and turns off its Failed LED (if present). It is then ready to commence VXIbus communications.
5. If the device requires further initialization, as in the case of a Message Based Device, its *Ready* bit remains cleared (to 0). When it is ready to accept its normal operational commands, it sets (to 1) its *Ready* bit.
6. A VMEbus Master may turn off a failed device's SYSFAIL* driver by setting (to 1) its *Sysfail Inhibit* bit. It may also force the device into a reset condition by setting its *Reset* bit.
7. When the *Reset* bit is cleared, the device begins another self test sequence.

A device may initialize its configuration registers any time within the 4.9 seconds after the de-assertion of SYSRESET*, typically in the following sequence.

1. At power-on SYSRESET* is asserted, forcing the device into a reset condition, disabling all VMEbus access to its configuration registers. The device asserts SYSFAIL* and lights its Failed LED (if present).
2. When SYSRESET* is de-asserted, the device begins initialization of its configuration registers. At this point VMEbus access remains disabled, SYSFAIL* remains asserted, and the Failed LED is lighted (if present).

3. After initializing to zero (0) the *Passed*, *Reset*, and *Sysfail Inhibit* bits and initializing to one (1) the *Ready* bit, the device may enable VMEbus access to its configuration registers.
4. If the device is unable to initialize the *Passed*, *Reset*, *Sysfail Inhibit*, and *Ready* bits, its configuration registers will remain inaccessible to the VMEbus.
5. After initializing all of its configuration registers, the device enables VMEbus access to its registers (if it hasn't already), clears the *Ready* bit, and begins its self test.
6. If the device fails to completely initialize its configuration registers, it enables VMEbus access to its registers (if it hasn't already), leaves its *Passed* bit cleared, its *Ready* bit set, SYSFAIL* asserted, and its Failed LED lighted (if present). After 4.9 seconds, these conditions indicate an initialization error.
7. When the device successfully completes both its configuration register initialization and its self test, it sets (to 1) its *Passed* bit, de-asserts SYSFAIL*, and turns off its Failed LED (if present). It is then ready to commence VXIbus communications.

C.2.1.2.2 Self Test Operation

The initialization sequence is characterized by eight states as illustrated in Figure C.4.

- Hard Reset: This is the state of the device when the SYSRESET* line is true. While in this state the device is inactive and its Status and Control registers may be initialized. The SYSFAIL* line is driven low, and the failed LED is lighted.
- Config Reg Init: During this optional state, the device configures its own configuration registers. This may include setting its Logical Address and its ID and Device Type register values, as well as initializing its Status, Control, and Offset registers. Initially, a device's configuration registers may be inaccessible via the VMEbus. In this state, the device continues to assert SYSFAIL* and its optional Failed LED is lighted. The config reg init state and any additional self test must complete within 4.9 seconds of SYSRESET*'s de-assertion or clearing of the *Reset* bit. If the configuration register initialization fails before the device enables VMEbus access to its configuration registers, the device remains in the CONFIG REG INIT state until the next Hard Reset. Before register access is enabled, the *Passed* bit is set to zero (0) and the *Ready* bit is set to one (1). The ID and Device Type registers may be initialized either before or after register access is enabled.
- Self Test: In the Self Test state, the device tests and initializes the functionality necessary for VXIbus communications. This test must complete within 4.9 seconds of SYSRESET*'s de-assertion or clearing of the *Reset* bit. If the self test fails, the device enters the failed state. If it passes, it enters the passed state. While in the self test state the device's VMEbus interface is active. However the device is incapable of responding to any commands other than *Reset* and *Sysfail Inhibit*. While in the SELF TEST state, the *Passed* bit remains cleared (to 0), the *Ready* bit is also cleared, and SYSFAIL* remains asserted.
- Init Failed: This state is entered if the configuration register initialization fails after the device has enabled access of its configuration registers. While in the INIT FAILED state, SYSFAIL* is asserted, the *Passed* bit is set to zero (0), and the *Ready* bit is set to one (1). This state may be exited by either a Hard Reset or a Soft Reset.

OBSERVATION C.2.14:

The output indications are identical for the CONFIG REG INIT and INIT FAILED states. Therefore a completely failed device that is incapable of even beginning to initialize its configuration registers will correctly indicate an init failed status.

- Failed: This state is entered if the self test fails. While in the FAILED state, the *Passed* bit and the *Ready* bit remain cleared (to 0) and SYSFAIL* remains asserted. This state may be exited by either a Hard Reset or a Soft Reset.

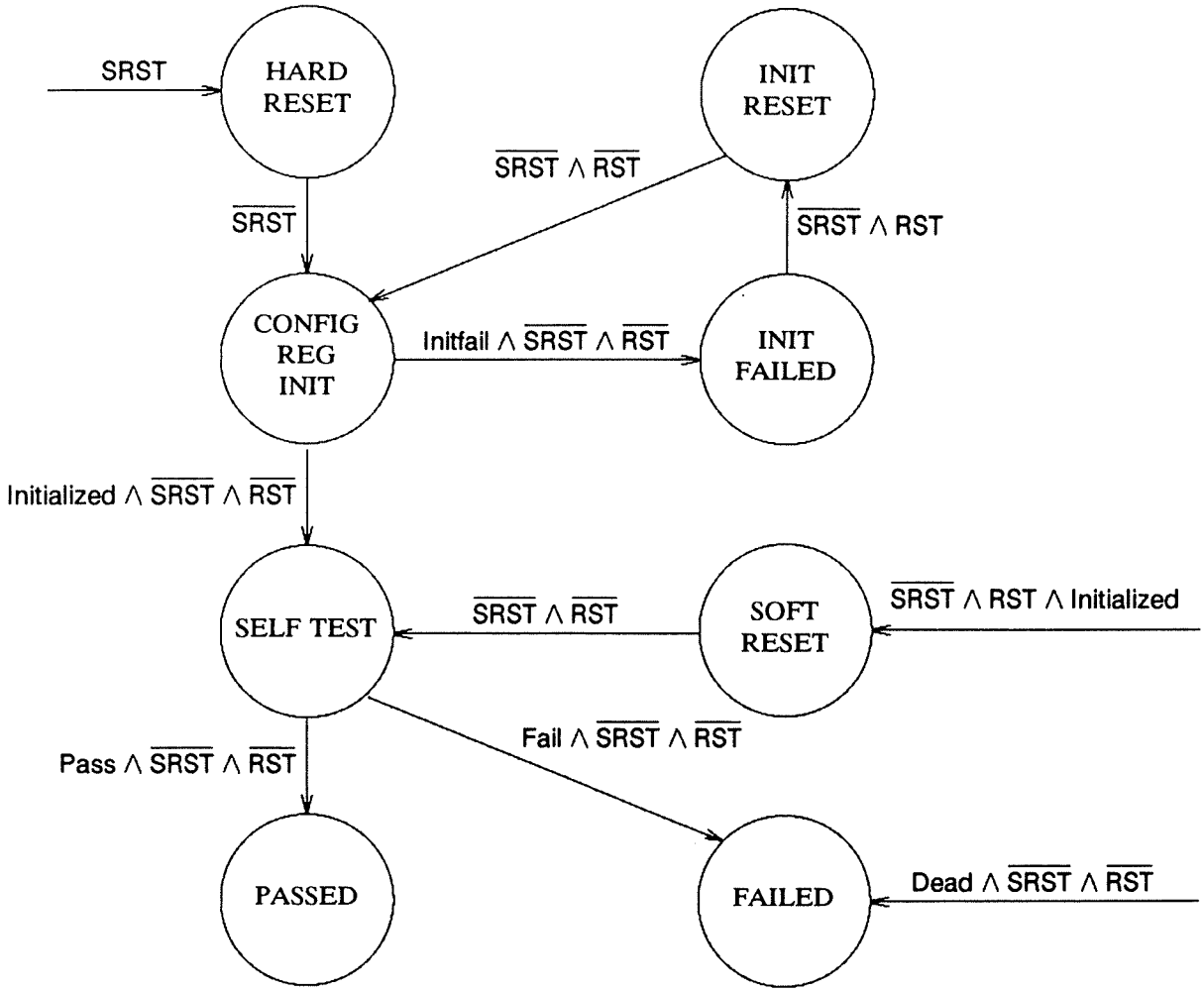


Figure C.4. Self Test State Diagram

OBSERVATION C.2.15:

The output indications are identical for the SOFT RESET, SELF TEST, and FAILED states. Therefore a completely failed device that is incapable of even beginning a self test will correctly indicate a failed status.

- Passed: This state is entered when the self test has successfully terminated. The SYSFAIL* line is de-asserted, and the *Passed* bit of the Status register is set (to 1). While in the PASSED state, the device is capable of fully supporting VXIbus communications.

OBSERVATION C.2.16:

The passed state can be further sub-divided in a device dependent manner. For Message Based Devices three sub-states are defined in Section C.2.4.4, "Message Based Device Operation".

- Soft Reset: This state is entered when the *Reset* bit of the Control register is set to one. While in this state a device is inactive, interrupts which are pending are unasserted, and all pending bus requests are removed. While in this state the device's VMEbus slave interface is active, however the device is incapable of responding to any commands other than *Reset* and *Sysfail Inhibit*. While in this state the Status register must accurately reflect the state of the device.

OBSERVATION C.2.17:

The VMEbus Specification (Section 3.3.2, "Requester") defines the protocols for proper removal of bus requests.

- Init Reset: This state is similar to the SOFT RESET state, except that the device's configuration registers have not been initialized (and therefore are meaningless).

The mnemonics used in the self test state diagram are listed in the following table.

SELF TEST Mnemonics

Mnemonic		Name	Type
SRST	=	SYSRESET* asserted (low)	VMEbus line
RST	=	Reset (1=true)	Control register bit
Fail	=	Self Test Failed	Internal signal
Pass	=	Self Test Passed	Internal signal
Dead	=	Device Failure	Internal signal
Initialized	=	Config Regs Initialized	Internal signal
Initfail	=	Init Config Regs Failed	Internal signal

OBSERVATION C.2.18:

The *Fail*, *Pass*, and *Dead* signals are generated by the Initialization process as sequencing signals to itself.

OBSERVATION C.2.19:

The *Dead* signal is generated by the device whenever it detects a catastrophic failure during normal operation.

The states of the initialization sequence produce the Status register indications given in the following table.

SELF TEST Status Bits

State	Status Register Bits	
	Passed	Ready
HARD RESET	(0)	(0/1)
CONFIG REG INIT	(0)	(1)
INIT FAILED	0	1
SELF TEST	0	0
FAILED	0	0
PASSED	1	0/1
SOFT RESET	0	0
INIT RESET	0	1

OBSERVATION C.2.20:

The Status register is inaccessible during the HARD RESET state. However, in many cases its bits will be initialized by SYSRESET* to indicate the device's state (CONFIG REG INIT or SELF TEST) immediately following HARD RESET.

OBSERVATION C.2.21:

The device's configuration registers may be inaccessible throughout the CONFIG REG INIT state. The Status register indications shown are meaningful only after VMEbus access has been enabled.

The status of the SYSFAIL* driver and the optional failed LED may be derived from the states of the *Passed* Status register bit and the *Sysfail Inhibit* Control register bit as described in the following table.

SELF TEST Indicators			
Register Bits		**Indicators**	
Passed Status Bit	Sysfail Inhibit Control Bit	SYSFAIL* Driver	FAILED LED
0	0	LOW	ON
0	1	HIGH	ON
1	X	HIGH	OFF

The VMEbus SYSRESET* line and the Control register interact to force the device into particular states as described in the following table.

SYSRESET* and Control Register Interactions			
Signal SYSRESET* Line	Reset Bit	**Current State**	**Effect**
LOW	X	X	HARD RESET state
↑	X	HARD RESET	Begin CONFIG REG INIT (if implemented) or SELF TEST
HIGH	1	SELF TEST, PASSED, or FAILED	SOFT RESET State
HIGH	1	INIT FAILED	INIT RESET State
HIGH	↓	SOFT RESET	Begin SELF TEST
HIGH	↓	INIT RESET	Begin CONFIG REG INIT

RULE C.2.16:
All devices implementing power-on self tests **SHALL** implement the HARD RESET, SELF TEST, PASSED, FAILED, and SOFT RESET states of the self test function.

RULE C.2.17:
Any device not implementing the initialization sequence **SHALL NOT** assert SYSFAIL* or clear (to 0) its *Passed* bit at any time.

RULE C.2.18:
A device **SHALL NOT** implement a CONFIG REG INIT and SELF TEST combination of more than 4.9 seconds total duration.

RULE C.2.19:
A device's ID and Device Type registers **SHALL** be valid before it enters the SELF TEST state.

PERMISSION C.2.6:
A device **MAY** execute self test operations while in the CONFIG REG INIT state, provided that its *Ready* bit is set (to 1), until its configuration registers have been initialized.

RULE C.2.20:
A device's VXIbus interface **SHALL** be fully operational before entering the PASSED state.

PERMISSION C.2.7:
A device **MAY** enter the FAILED state any time it detects a catastrophic failure during normal operation.

OBSERVATION C.2.22:
The *Sysfail Inhibit* bit of the Control register is used to disable the device's SYSFAIL* driver.

OBSERVATION C.2.23:
The Failed LED is optional. Other LED's such as a SYSFAIL* inhibit indicator may also be optionally implemented.

OBSERVATION C.2.24:
Once a device has enabled access of its configuration registers, they will remain accessible until the next Hard Reset.

RULE C.2.21:
Once a device enters either the INIT FAILED or SELF TEST state, it **SHALL NOT** change its Logical Address except as provided for by Dynamic Configuration (see Section F, "Dynamic Configuration"), until the next Hard Reset.

OBSERVATION C.2.25:
A Soft Reset will not affect a device's Logical Address, configuration register accessibility, Offset register, or *A24/A32 Active* bit.

RULE C.2.22:
After either a Hard Reset or a Soft Reset, a device **SHALL** remain deactivated until activated by its Commander.

RULE C.2.23:
A deactivated device **SHALL NOT** assert any VMEbus IRQ lines or initiate any data transfer cycles.

OBSERVATION C.2.26:
A Commander activates a device in a device dependent manner. A standard method of activating Message Based Devices is given in Section C.2.4.5, "Message Based Device Configuration".

RULE C.2.24:
A VXIbus device, other than a Resource Manager (see Section C.4.1, "Resource Manager"), **SHALL NOT** autonomously activate itself.

C.2.1.3 PRIORITY INTERRUPTS

Use of priority interrupts is optional for VXIbus devices. However all VXIbus devices recognize a standard format for the STATUS/ID word provided in response to interrupt acknowledge cycles.

OBSERVATION C.2.27:
Priority interrupts provide a mechanism for routing information between a device with an Interrupter and a device with an Interrupt Handler (on the same IRQ line). The Interrupt Handler might not be the intended recipient of the information. If an Interrupt Handler receives an interrupt which it recognizes to be intended for another device, it may forward the information carried by the interrupt, to the intended recipient via either a Signal or another interrupt.

C.2.1.3.1 Interrupters

RULE C.2.25:
VXIbus devices with Interrupter capability **SHALL** allow the user to configure which Interrupt Request line (IRQ1*-IRQ7*) is used.

RECOMMENDATION C.2.5:
VXIbus Interrupters should implement Release On Acknowledge (ROAK).

OBSERVATION C.2.28:
Message Based Devices that are Interrupters are required to implement ROAK. See Section C.2.4, "Message Based Devices".

RULE C.2.26:
Interrupters which implement D16 STATUS/ID Transfer capabilities **SHALL** also implement D08(O) STATUS/ID Transfer capabilities.

RULE C.2.27:
Interrupters which implement D32 STATUS/ID Transfer capabilities **SHALL** also implement D16 STATUS/ID Transfer capabilities and D08(O) STATUS/ID Transfer capabilities.

RULE C.2.28:
VXIbus devices **SHALL** implement combined D08(O), D16, and D32 interrupt response modes such that the size of the STATUS/ID word is selected according to the type of interrupt acknowledge cycle detected, according to the following table.

STATUS/ID Word Size			
Cycle Type	Combined Mode		
	D08	D08/16	D08/16/32
8 Bit	8	8	8
16 Bit	8	16	16
32 Bit	8	16	32

OBSERVATION C.2.29:
The VMEbus specification requires that D08(O) Interrupters respond to 8, 16, and 32 bit interrupt acknowledge cycles with an 8 bit STATUS/ID word. D16 Interrupters respond to 16 and 32 bit cycles with a 16 bit word.

OBSERVATION C.2.30:
The preceding three rules guarantee that no matter what width STATUS/ID read the Interrupt Handler generates, the Interrupter will respond with the appropriate width of STATUS/ID.

OBSERVATION C.2.31:
Many of the standard Interrupt Handlers which exist today (1987) generate only D08(O) STATUS/ID read cycles. The preceding three rules guarantee that Interrupters which respond to D16 and D32 STATUS/ID cycles will also be able to respond to Interrupt Handlers which request D08(O) responses.

RULE C.2.29:
All VXIbus Interrupters **SHALL** provide STATUS/ID responses to interrupt acknowledge cycles according to the format defined in the following table.

Bit #	31 ← 16	15 ← 8	7 ← 0
Contents	D32 Extension	Cause/ Status	Logical Address

The fields of the STATUS/ID word are defined below.

- D32 Extension: These optional bits may be provided by a D32 Interrupter in order to give additional information about its status or the cause of the interrupt. The value $FFFF_{16}$ is reserved to mean *No Extension Given*. Otherwise the format of this field is unspecified.

  OBSERVATION C.2.32:
  If a D08 (or D16) Interrupt Handler reads a 32 bit STATUS/ID word, the upper 24 (or 16) bits of data will be lost.

- Cause/Status: These bits give the cause of the interrupt and/or reflect the Interrupter's Status register. The value FF_{16} is reserved to mean *No Cause/Status Given*.

  OBSERVATION C.2.33:
  If a D08 Interrupt Handler reads a 16 bit STATUS/ID word, the upper 8 bits of data will be lost.

- Logical Address: This field contains the Logical Address of the Interrupter device.

RECOMMENDATION C.2.6:
A VXIbus Interrupter should provide a means of masking VMEbus interrupts.

C.2.1.3.2 Interrupt Handlers

RULE C.2.30:
VXIbus devices with Interrupt Handler capability **SHALL** allow the user to configure which Interrupt Request line (IRQ1*-IRQ7*) is monitored.

RECOMMENDATION C.2.7:
VXIbus Interrupt Handlers should implement D16 or D32 interrupt acknowledge cycles.

SUGGESTION C.2.2:
For compatibility with Non-VXIbus Interrupters, design VXIbus Interrupt Handlers to include D08(O) interrupt acknowledge cycle generation capability.

C.2.1.4 VMEbus MASTER CAPABILITIES

As discussed in the specific device class sections, Master capability is optional for Register Based and Extended VXIbus Devices, prohibited for Memory Devices, and required for some Message Based Devices. The capabilities are intended to ensure compatibility with all VXIbus slaves, as well as with many non-VXIbus slaves.

C.2.1.4.1 Bus Requesters

VXIbus devices implement a fair requester protocol which requires that a level *n* requester, after receiving a bus grant via BG*n*IN*, may not request the bus again until at least 30 ns after the next low to high transition of Bus Request line BR*n**. This forces a Round Robin type of bus management on a single Bus Request/Grant level. This protocol is also known as Request on No Request (RONR).

RULE C.2.31:
All VXIbus devices with Master capability **SHALL** implement the Fair Requester protocol.

OBSERVATION C.2.34:
The above rule provides a more egalitarian environment than the standard VMEbus bus request protocols. This matches instrumentation requirements better than the default prioritization of devices provided by the daisy chains.

RULE C.2.32:
VXIbus devices with Master capability **SHALL** use Bus Request/Grant level 3 as a default.

RECOMMENDATION C.2.8:
VXIbus devices with Master capability should allow the user to configure which Bus Request/Grant level is used.

PERMISSION C.2.8:
VXIbus devices with Master capability **MAY** provide the capability of selecting another type of Requester.

OBSERVATION C.2.35:
The above permission allows designers of high performance boards with real time constraints to implement bus locking. Not implementing a Fair Requester can cause system performance difficulties.

RECOMMENDATION C.2.9:
IF a device has the Fair Requester disabled
THEN it should be moved to another Bus Request/Grant level.

RECOMMENDATION C.2.10:
A VMEbus Master should implement cycle by cycle arbitration. A Master should not hold the bus while doing computation. In the case of a read from a device, a subsequent operation, and then write to another device, the Master should allow arbitration of the VMEbus after the initial read.

PERMISSION C.2.9:
A VMEbus Master **MAY** maintain control of the data transfer bus for the time required to complete a Block Transfer Cycle.

OBSERVATION C.2.36:
The VMEbus Specification allows for the possibility of a Master holding the data transfer bus (DTB) indefinitely.

C.2.1.5 TERMINATING OPERATION

A functioning VXIbus device terminates its operation in an orderly and consistent manner. This allows for putting devices off-line without adversely affecting other operating devices. It also allows for efficient reconfiguration of the VXIbus hierarchy. Four different methods are available to a Commander or a Resource Manager for terminating the operations of a device.

- Hard Reset: When the SYSRESET* line goes true all devices immediately terminate their operation. Hard Reset is used only when it is necessary to promptly terminate the operation of all devices in a VXIbus system. A Hard Reset will result in the system being configured by the Resource Manager.

RECOMMENDATION C.2.11:
Asserting SYSRESET* during operation is a drastic measure and should only be initiated to emulate a power-up cycle.

- Soft Reset: A Commander may terminate the operation of one of its Servants by setting to one (1) the *Reset* bit in the Servant's Control register.

RULE C.2.33:
A Commander **SHALL NOT** Soft Reset any device not in its Servant list.

PERMISSION C.2.10:
The Resource Manager **MAY** Soft Reset a Top Level device.

- VXIbus commands: A Commander may terminate the operation of a Message Based Servant by sending it one of a number of low-level VXIbus commands. Refer to Section C.2.4.7, "Terminating Operation", for a detailed description of these commands.

RULE C.2.34:
Only a device's Commander may terminate the operation of that device by sending it a VXIbus device termination command.

- Device dependent termination: A Commander may terminate the operation of a non Message Based Device in a device dependent manner, known to that Commander. The Servant ends up in a well-defined state (refer to Section C.2.2.4, "Terminating Operation").

RULE C.2.35:
A Commander **SHALL NOT** use a device dependent method to terminate the operation of a Message Based Device.

RULE C.2.36:
When a device is terminated by a Soft Reset, a VXIbus command or in a device dependent manner, it **SHALL** retain the contents of its ID register, Device Type register, Offset register, Logical Address, and the state of its A24/A32 enable bit.

C.2.2 Register Based Devices

A Register Based Device is generally a slave only device. Communication with one of these devices is usually accomplished via reads and writes of its registers. However, Register Based Devices may also utilize interrupts. One example of this class of device is a relay multiplexor device which is switched by writes to specific device registers. Register Based Devices are the simplest VXIbus devices, and are suitable for low cost implementations.

C.2.2.1 DATA TRANSFER CAPABILITIES

OBSERVATION C.2.37:

The VXIbus protocols provide no standard mechanism for identifying Register Based Devices with Master capability, or for the regulation of the address space accessed by such devices. It is recognized that limited Master capability may in some cases be important for Register Based Servants. However, a Register Based Device with Master capability could make errant writes into unprotected address spaces unless such accesses are very closely guarded (such as in the case of a DMA capable disc interface). Care should be exercised to prevent the autonomous operation of a Register Based Device outside of the limits defined by its Commander.

C.2.2.2 PRIORITY INTERRUPTS

Register Based Devices are not required to use VMEbus interrupts. Some Register Based Devices may use interrupts for status reporting.

RECOMMENDATION C.2.12:

IF a Register Based Device has a status register bit to indicate a busy or not ready state,
AND that bit can normally be asserted for periods longer than fifty (50) microseconds,
THEN the device should have interrupt capability that may be enabled to be asserted at the end of the busy/not ready period.

OBSERVATION C.2.38:

The preceding recommendation is intended to minimize backplane traffic due to excessive polling of Status registers.

OBSERVATION C.2.39:

A Register Based Interrupter responds to interrupt acknowledge cycles with a STATUS/ID word conforming to the standard VXIbus format defined in Section C.2.1.3.1 "Interrupters".

The standard VXIbus STATUS/ID word format is illustrated in the following table.

Bit #	31 ← 16	15 ← 8	7 ← 0
Contents	D32 Extension	Cause/ Status	Logical Address

OBSERVATION C.2.40:

The Cause/Status and D32 Extension fields of the STATUS/ID word are optional. Their bit definitions are entirely device dependent for Register Based Devices.

C.2.2.3 REGISTER BASED DEVICE REGISTERS

Register Based Devices have both VXIbus standard configuration registers and device dependent operational registers. The location and quantities of these registers are determined from the device's Logical Address, and other information stored by its configuration registers (See Section C.2.1.1.2, "Configuration Registers"). The *Address Space* field of the ID register defines the location of the operational registers as follows.

A16 Only: The operational registers occupy addresses 06_{16} - $3F_{16}$ relative to the device's A16 base address.

A16/A24: The operational registers occupy addresses 08_{16} - $3F_{16}$ relative to the device's A16 base address and a block of A24 addresses located by the device's Offset register

A16/A32: The operational registers occupy addresses 08_{16} - $3F_{16}$ relative to the device's A16 base address and a block of A32 addresses located by the device's Offset register

OBSERVATION C.2.41:
All Register Based Device operational registers have device dependent functionality.

C.2.2.4 TERMINATING OPERATION

Three different methods are available for terminating the operation of a Register Based Device. These are Hard Reset, Soft Reset and Device Dependent Termination. The orderly termination of device operation allows for deactivating a device without adversely affecting other operating devices. It also allows for efficient reconfiguration of the VXIbus hierarchy.

OBSERVATION C.2.42:
When a VXIbus Register Based Device is deactivated it may not assert any VMEbus IRQ lines or intiate any data transfers.

C.2.2.4.1 Hard Reset

Upon receipt of a Hard Reset (assertion of SYSRESET*), a Register Based Device enters, as does any other device class, the HARD RESET state. From that state a device continues its initialization until it ends in one of the possible states as defined in Section C.2.1.2, "Device Initialization and Diagnostics". Hard Reset is used only when it is necessary to promptly terminate operation of all devices in a VXIbus system. A Hard Reset always affects the entire VXIbus system, and will result in the system being configured by the Resource Manager.

C.2.2.4.2 Soft Reset

A Commander may terminate the operation of a Register Based Servant by setting to one the Reset bit in the Servant's Control register. When the Commander clears the Reset bit, a Register Based Device enters, as does any other device class, the SELF TEST state (refer to Section C.2.1.2, "Device Initialization and Diagnostics"). If successful, the device will then enter the PASSED state. At this point, the device will be in a benign state with no links to any other devices.

OBSERVATION C.2.43:
A Soft Reset is the only non device dependent method available to a Commander for terminating the operation of a Register Based Servant.

OBSERVATION C.2.44:
A Register Based Device that has been Soft Reset may be inconsistent with the remainder of the system. It is important that such a device not be activated until all inconsistencies have been resolved.

RULE C.2.37:
Before a Commander reactivates a device after having Soft Reset that device, the Commander **SHALL** guarantee that the device is in a state consistent with the rest of the system.

OBSERVATION C.2.45:
A Register Based Device which has been Soft Reset is activated by its Commander in a device dependent manner.

RULE C.2.38:
A Register Based Device which has been Soft Reset **SHALL** retain its assigned Logical Address if it has been dynamically configured (refer to Section F, "Dynamic Configuration"). It **SHALL** also retain any A24/A32 memory assignment.

C.2.2.4.3 Device Dependent Termination

Soft Resetting a device will cause it to go through its self test. It may be desirable for a Commander to simply deactivate a Register Based Device without having to go through self test.

PERMISSION C.2.11:
A Register Based Device **MAY** have a device dependent method other than Soft Reset that allows the device's Commander to terminate the device's operation.

OBSERVATION C.2.46:
A Register Based Device which has been deactivated in a device dependent manner is also reactivated by its Commander in a device dependent manner.

RULE C.2.39:
Before a Commander reactivates a device after having terminated its operation in a device dependent manner, the Commander **SHALL** guarantee that the device is in a state consistent with the rest of the system.

C.2.3 Memory Devices

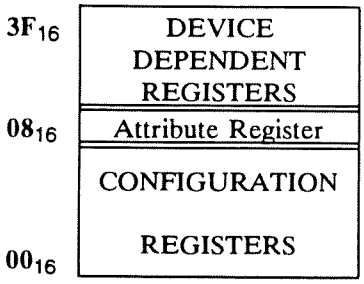
VXIbus Memory Devices provide either permanent or temporary data storage in the form of blocks of ROM, RAM, etc. in the VMEbus A24 or A32 address space. The addresses of these memory blocks are configured via the VXIbus configuration registers.

C.2.3.1 MEMORY DEVICE REGISTERS

A Memory Device has both A16 configuration registers and A24 or A32 operational registers. In addition it has an ATTRIBUTE REGISTER, which indicates the nature of its operational memory registers.

C.2.3.1.1 Memory Device Attribute Register

The attribute register is a read only register which lists important characteristics of the Memory Device. The following diagram contains the configuration register map for a Memory Device.



The Attribute Register has the following format.

Attribute Register:

Bit #	15 - 14	13	12	11	10 - 8	7	6 - 4	3 - 0
Contents	Memory Type	N/S	BT*	N_P*	Access Speed	D32*	RESERVED	Device Dependent

- Memory Type: These two bits contain information about the type of memory contained in this device.

BIT #		Meaning
15	14	
1	1	RAM
0	1	ROM
1	0	OTHER
0	0	RESERVED

- N/S: A one in this position indicates that the device is accessible in both Non-Privileged and Supervisory modes. A zero in this position indicates that the device is accessible only in Supervisory mode.
- BT*: A zero in this position indicates that the device has Block Transfer Capability.

- N_P*: The meaning of this field is dependent on the *memory type* field. If the memory type is RAM (11), a zero (0) in this field indicates that the RAM is non-volatile. If the memory type is ROM (01), a zero (0) in this field indicates that the device is electrically programmable.
- Access Speed: The value in this field indicates the memory access times, as listed in the following table.

BIT # 10	9	8	Access Time (t)
0	0	0	0 ns $\leq t <$ 50 ns
0	0	1	50 ns $\leq t <$ 100 ns
0	1	0	100 ns $\leq t <$ 200 ns
0	1	1	200 ns $\leq t <$ 400 ns
1	0	0	400 ns $\leq t <$ 800 ns
1	0	1	800 ns $\leq t <$ 1600 ns
1	1	0	1600 ns $\leq t$
1	1	1	Device Dependent

OBSERVATION C.2.47:
The access time is the maximum delay from the assertion DS0* or DS1* until the MEMORY device asserts DTACK*.

- D32*: A zero in this position indicates that the device has D32 capability in addition to its D16 and/or D08.
- RESERVED: Reserved for future definition and must default to all ones (7_{16}).
- Device Dependent: These bits may be defined by the device's manufacturer.

A write to the Attributes Register location has no effect. Its use is reserved for VXIbus definition.

C.2.3.1.2 Memory Device Operational Registers

A Memory Device's operational registers are its functional memory locations. The location and quantities of these registers are determined from information stored in the device's configuration registers (See Section C.2.1.1.2, "Configuration Registers"). The *Address Space* field of the ID register defines the location of the operational registers as follows.

A16 Only: This configuration is undefined for Memory devices.

A16/A24: The operational registers occupy a block of A24 addresses located by the device's Offset register.

A16/A32: The operational registers occupy a block of A32 addresses located by the device's Offset register.

PERMISSION C.2.12:
A Memory Device's A16 device dependent registers **MAY** be used for device specific purposes.

C.2.4 Message Based Devices

Message Based Devices support the VXIbus configuration and communication protocols. All Message Based Devices are capable of at least a minimum set of standard communication protocols. Examples of Message Based Devices are any sophisticated devices with local intelligence that require a level of communication capability such as DMM's, Spectrum Analyzers, Display Controllers, 488-VXIbus interface devices, Switch controllers, etc.

C.2.4.1 DATA TRANSFER CAPABILITIES

Message Based Devices utilize a standard set of registers to implement the VXIbus messaging protocols. These registers operate in the A16 address space and require D16 capability.

RULE C.2.40:
All Message Based Commanders **SHALL** have A16 and D16 Master capability.

RECOMMENDATION C.2.13:
All Message Based Devices should have A16 and D16 Master capability.

OBSERVATION C.2.48:
Servant-only devices with A16 Master capability are able to send status messages to their associated Commanders efficiently, without the use of priority interrupts.

RECOMMENDATION C.2.14:
All Message Based Devices should have A24 Master capability.

RULE C.2.41:
All Message Based A16 Masters **SHALL** be capable of accessing the entire A16 address range.

RULE C.2.42:
All Message Based A24 Masters **SHALL** be capable of accessing the entire address range from 200000_{16} through $DFFFFF_{16}$.

PERMISSION C.2.13:
A Message Based A24 Master **MAY** be capable of accessing the entire A24 address range from 000000_{16} through $FFFFFF_{16}$.

SUGGESTION C.2.3:
Message Based Devices should have D08(EO) Master capability.

OBSERVATION C.2.49:
D08(EO) Master capability facilitates compatibility with a broader range of Non-VXIbus devices than would otherwise be possible.

PERMISSION C.2.14:
Message Based Devices **MAY** implement A32 and/or D32 capabilities.

OBSERVATION C.2.50:
A32 Master capability is required for control of any A32 Register Based or Memory devices or any A32 Non-VXIbus devices.

RULE C.2.43:
All Message Based A32 Masters **SHALL** be capable of accessing the entire A32 address range from 20000000_{16} through $DFFFFFFF_{16}$.

PERMISSION C.2.15:

A Message Based A32 Master **MAY** be capable of accessing the entire A32 address range from 00000000_{16} through $FFFFFFFF_{16}$.

OBSERVATION C.2.51:

VXIbus protocols support the inclusion of A32 or A24 RAM in Message Based Devices.

The preceding rules, etc. are summarized in the following table.

	A16	A24	A32	D08(EO)	D16	D32
MASTER	S/R	REC	MAY	SUG.	S/R	MAY
SLAVE	SHALL	MAY	MAY	MAY	SHALL	MAY

SUG. - Suggested.
REC - Recommended.
S/R - SHALL for Commanders. Recommended for others.

C.2.4.2 PRIORITY INTERRUPTS

RECOMMENDATION C.2.15:

A Message Based Device should implement either VMEbus Master or Interrupter capability.

Interrupt Handler capability is entirely optional.

OBSERVATION C.2.52:

The *Asynchronous Mode Control* command may be used to select and enable the transmission mode (interrupt or signal) of the different classes (response and event) of interrupts.

RULE C.2.44:

Message Based devices with Interrupter capability **SHALL** implement Release On Acknowledge (ROAK).

Three word serial commands are defined to allow a device's Commander to query and modify the interrupt lines selected by a Message Based Device with Interrupter capability. These commands are:

Read Interrupters - This command is used to read the number of interrupt lines the device may drive simultaneously, i.e. the number of Interrupters in the device.

Read Interrupter Line - This command is used to read which VMEbus interrupt line is driven by a particular Interrupter in the device.

Assign Interrupt Line - This command is used to assign one of the device's Interrupters to a particular VMEbus interrupt line.

Within a device, multiple Interrupters are identified in some device dependent consecutive order, starting with #1. They are designated *Interrupter #1, Interrupter #2*, etc.

RECOMMENDATION C.2.16:

A Message Based Device with Interrupter capability should implement Programmable Interrupter (PI) capability, which provides a mechanism to program and read back each interrupt line connection to that device's Interrupter(s). This is achieved by supporting the *Read Interrupters*, *Read Interrupter Line*, and *Assign Interrupter Line* commands.

PERMISSION C.2.16:

Message Based Commanders **MAY** implement Interrupt Handler capability.

Three word serial commands are defined to allow a device's Commander to query and modify the interrupt lines selected by a Message Based Device with Interrupt Handler capability. These commands are:

Read Handlers - This command is used to read the number of interrupt lines the device may handle simultaneously, i.e. the number of Interrupt Handlers in the device.

Read Handler Line - This command is used to read which VMEbus interrupt line is handled by a particular Interrupt Handler in the device.

Assign Handler Line - This command is used to assign one of the device's Interrupt Handlers to a particular VMEbus interrupt line.

Within a device, multiple Interrupt Handlers are identified in some device dependent consecutive order, starting with #1. They are designated *Handler #1, Handler #2*, etc.

RECOMMENDATION C.2.17:

A Message Based Device with Interrupt Handler capability should implement Programmable Handler (PH) capability, which provides a mechanism to program and read back each interrupt line connection to that device's Interrupt Handler(s). This is achieved by supporting the *Read Handlers*, *Read Handler Line*, and *Assign Handler Line* commands.

RULE C.2.45:

IF a Message Based Device implements Interrupter capability
THEN it **SHALL** respond to an interrupt acknowledge with the 16 bit Status/ID word defined in the following tables.

There are two different formats for the STATUS/ID word, distinguished by the value of its bit #15.

Bit #	31 ← 16	15	14 ← 8	7 ← 0
Contents	D32 Extension	0	Response	Logical Address

Bit #	31 ← 16	15	14 ← 8	7 ← 0
Contents	D32 Extension	1	Event	Logical Address

The fields of the STATUS/ID word are defined below.

- D32 Extension: The contents of this optional field are device dependent.

  ### OBSERVATION C.2.53:

  D08(O) and D16 STATUS/ID generators will always return $FFFF_{16}$ in this field.

- Response: This field is identical to bits 14 ← 8 of the device's Response register as defined in Section C.2.4.3.1, "Message Based Device Communication Registers".

  ### OBSERVATION C.2.54:

  A Message Based Device will not generate any response signals or response interrupts unless such response signals or response interrupts have been enabled by the *Control Response* command.

OBSERVATION C.2.55:
Devices which are not capable of generating response signals or response interrupts are not required to support the *Control Response* command.

- Event: This field indicates the *event* associated with the interrupt. See Section E.4, "Protocol Events", for a listing of the defined events.

OBSERVATION C.2.56:
D08(O) STATUS/ID generators will always return FF_{16} in bits 15 ← 8.

- Logical Address: This field contains the Logical Address of this device.

OBSERVATION C.2.57:
The VXIbus provides alternate mechanisms for transmitting responses and events: interrupts and signals (See the Signal register definition in Section C.2.4.3.1, "Message Based Device Communication Registers"). Non-master devices always transmit via interrupts, while master devices may send signals. Some devices are capable of receiving only signals, whereas others may be only Interrupt Handlers. The VMEbus provides only seven (7) interrupt lines. In a large system these lines may be overloaded. Such a system may provide a mechanism for converting signals to interrupts, converting interrupts to signals, or even translating from one interrupt line to another.

C.2.4.3 MESSAGE BASED DEVICE REGISTERS

Message Based Devices have both VXIbus standard configuration registers and operational registers. The location and quantities of these registers are determined from the device's Logical Address, and other information stored by its configuration registers (See Section C.2.1.1.2, "Configuration Registers"). The *Address Space* field of the ID register defines the location of the operational registers as follows.

A16 Only: The operational registers occupy addresses 06_{16} → $3F_{16}$ relative to the device's A16 base address.

A16/A24: The operational registers occupy addresses 08_{16} → $3F_{16}$ relative to the device's A16 base address and a block of A24 addresses located by the device's Offset register

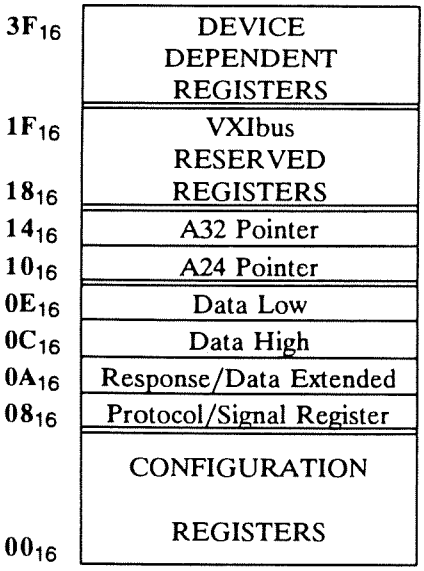
A16/A32: The operational registers occupy addresses 08_{16} → $3F_{16}$ relative to the device's A16 base address and a block of A32 addresses located by the device's Offset register

OBSERVATION C.2.58:
All Message Based Device A24 and A32 operational registers have device dependent functionality.

C.2.4.3.1 Message Based Device Communication Registers

Message Based Devices implement a standard set of A16 operational registers, known as communication registers. These registers are shown in the following table. The addresses are relative to a device's A16 base address. A block of registers is reserved for use by future optional protocols. It is anticipated that the definition of these registers will be tied to the definition of the reserved bits in the Protocol register.

The communication registers are organized as follows:



RULE C.2.46:

All Message Based Devices **SHALL** implement the Protocol, Response, and Data Low registers.

OBSERVATION C.2.59:

The Signal, Data Extended, Data High, A24 Pointer, and A32 Pointer registers are optional.

The communication registers are defined as follows:

Protocol Register: The read contents of this 16 bit register indicate which protocols the device supports, and indicate additional communication capabilities of the device.

Bit #	15	14	13	12	11	10	9 - 4	3 - 0
Contents	CMDR*	Signal Register*	Master*	Interrupter	FHS*	Shared Memory*	RESERVED	Dev Dep

- CMDR*: A one (1) in this field indicates that the device has Servant only capability. A zero (0) in this field indicates that the device has both Servant and Commander capability.
- Signal Register*: A zero (0) in this field indicates that the device has a Signal register. See below for a definition of a Signal register.
- Master*: A zero (0) in this field indicates that the device has VMEbus Master capability.
- Interrupter: A one (1) in this field indicates that the device has Interrupter capability.
- FHS*: A zero (0) in this field indicates that this device's data registers support the Fast Handshake mode (See Section C.3.3.2, "Fast Handshake Transfers"). A one (1) indicates that only the normal transfer mode is implemented.
- Shared Memory*: A zero (0) in this field indicates that this device supports the optional shared memory protocol (See Section G, "Shared Memory Protocols") and implements one or both of the A24 Pointer and A32 Pointer registers. A one (1) indicates that the shared memory protocol is not supported.

- RESERVED: Reserved for future definition and must default to all ones ($3F_{16}$).
- Dev Dep: These device dependent bits may be defined by the device's manufacturer.

Signal Register: A write to this 16 bit register is used for general device to device signaling. For efficient system operation the Commander must be capable of detecting and responding quickly to any Signal register write. A signal consists of the Logical Address of the sender and other signal specific information.

There are two different formats for the signals, distinguished by the value of its bit #15.

Bit #	15	14 ← 8	7 ← 0
Contents	0	Response	Logical Address

Bit #	15	14 ← 8	7 ← 0
Contents	1	Event	Logical Address

The fields of the signal are defined below.

- Response: This field is identical to bits 14 ← 8 of the device's Response register as defined below.
- Event: This field indicates the *event* associated with the signal.

OBSERVATION C.2.60:
Response signals are generally used to respond to a request made by a device. Event signals are generally asynchronous events analogous to interrupts. The *Asynchronous Mode Control* command may be used to select and enable the transmission mode (interrupt or signal) of the different classes (response and event) of signals and interrupts.

RULE C.2.47:
IF a Message Based Device implements a functional Signal register,
THEN there **SHALL** be a Write mode Location Monitor at the Signal register address.

RULE C.2.48:
IF a Message Based Device supports D08(EO) writes to its Signal register,
THEN its Signal register D08 Location Monitor **SHALL** detect odd byte accesses only.

RULE C.2.49:
IF a write to a device's Signal register would result in loss of data,
THEN that device **SHALL** respond to such a write by asserting BERR* instead of DTACK*.

OBSERVATION C.2.61:
A loss of data would occur if a signal write were to overwrite a previous signal that had not yet been read by the device's internal microprocessor.

OBSERVATION C.2.62:
A device may delay the DTACK* or BERR* handshake up to 20 μs (see RULE B.2.1) in an effort to prevent data loss.

OBSERVATION C.2.63:
If a write to a device's Signal register results in a bus error, then that signal will not be received by the device. In such a case the sender may choose to retransmit the signal.

PERMISSION C.2.17:
A Message Based Device **MAY** implement a non-functional Signal register. This is indicated by a "1" in bit 14 of the device's Protocol register. In this case the device may respond with DTACK* to Signal register writes, even though it stores no data from that write cycle.

RECOMMENDATION C.2.18:
If a Commander is intended to support a variety of Servants and configurations, it should implement a functional Signal register and the receipt of signals.

Response Register: A read of this 16 bit register returns the status of the device's communication registers and their associated functions.

The Message Based device's Response register indicates critical device status information. Each bit indicates a single status item as follows:

Bit #	15	14	13	12	11	10	9	8	7	6 ← 0
Contents	0	RESERVED	DOR	DIR	Err*	Read Ready	Write Ready	FHS Active*	Locked*	Device Dependent

The fields of the Response register are defined below.

- Bit 15 is always zero.

  ### OBSERVATION C.2.64:
  Bit 15 is defined to be zero in order to provide consistency with the format of signals and Interrupter STATUS/ID words.

- RESERVED: This bit is reserved for future VXIbus definition. Its value is always one (1).

- DOR (Data Out Ready) [optional]: A one indicates that the device is ready to output data to its Commander. Its operation is described in Section C.3.3.3, "Byte Transfer Protocol".

  ### OBSERVATION C.2.65:
  Devices that support the *Byte Request* command are required to implement the *DOR* bit.

- DIR (Data In Ready) [optional]: A one indicates that the device is ready to accept data from its Commander. Its operation is described in Section C.3.3.3, "Byte Transfer Protocol".

  ### OBSERVATION C.2.66:
  Devices that support the *Byte Available* command are required to implement the *DIR* bit.

- Err* (Error) [required]: A zero (0) indicates that an error has occurred in one of the serial protocols (Word Serial, Longword Serial, or Extended Longword Serial. See Section C.3.3.1, "Serial Protocols".), and has not yet been reported. A one (1) indicates that all such codes have been reported via the *Read Protocol Error* command. See Section C.3.3.4, "Error Handling".

  ### OBSERVATION C.2.67:
  The *Read Protocol Error* command is used to report the reason for an assertion of the *Err** bit.

- Read Ready [Required]: A one indicates that its data registers contain data to be read. This bit is set when data from internal operations is available in the Data Low and Data High registers. It is cleared to zero by a read of the Data Low register.

RULE C.2.50:
The *Read Ready* bit **SHALL** be cleared before DTACK* is released during any data transfer cycle that reads the Data Low register. A device **SHALL NOT** clear its *Read Ready* bit at any other time, except as outlined in Table C.2.

- Write Ready [Required]: A one indicates that this device is ready for data transfers to its data registers. This bit is set when the device is ready for a write to any of the data registers. It is cleared by a write to the Data Low register.

RULE C.2.51:
The *Write Ready* bit **SHALL** be cleared before DTACK* is released during any data transfer cycle that writes to the Data Low register. A device **SHALL NOT** clear its *Write Ready* bit at any other time.

- FHS Active* [optional]: A zero indicates that Fast Handshakes are currently enabled for this device's Servant registers (See Section C.3.3.2, "Fast Handshake Transfers"). This bit is cleared (to 0) whenever the Fast Handshake Mode is entered, and set(to 1) any time the device exits the Fast Handshake Mode.
- Locked* [optional]: A zero indicates that the Commander of this device has locked access to it from other local sources. (e.g., local lockout of IEEE-488)
- Device Dependent: The bits in this field are available to the device designer for device and/or application specific status information.

RULE C.2.52:
Each Message Based device **SHALL** be implemented in such a manner that each of the Response register bits 13 ← 8 maintains its current state whenever the device modifies any other Response register bit(s).

OBSERVATION C.2.68:
The preceding rule cannot be satisfied by an implementation which requires a read-mask-write sequence to update the Response register, if that sequence allows an intervening access of the Data Low register (which will clear either the *Read Ready* or *Write Ready* bit) by another device.

OBSERVATION C.2.69:
Transitions of any of bits 14 ← 8 of the Response register may be enabled to generate response interrupts or response signals as defined in Section C.2.4.2, "Priority Interrupts".

Data Extended Register: A write to this optional register is the most significant word of input data or a command.

PERMISSION C.2.18:
A Message Based Device **MAY** implement a non-functional Data Extended register.

RULE C.2.53:
IF a Message Based Device implements a functional Data Extended register,
THEN there **SHALL** be a Write mode Location Monitor at the Data Extended register address.

RULE C.2.54:
IF a Message Based Device supports D08(EO) writes to its Data Extended register,
THEN its Data Extended register D08 Location Monitor **SHALL** detect odd byte accesses only.

Data High Register: A write to this optional register is the second least significant word of write data. A read of this register is the most significant word of read data.

PERMISSION C.2.19:
A Message Based Device **MAY** implement a non-functional Data High register.

RULE C.2.55:
IF a Message Based Device implements a functional Data High register,
THEN there **SHALL** be a Write mode Location Monitor at the Data High register's address.

RULE C.2.56:
IF a Message Based Device supports D08(EO) writes to its Data High register,
THEN its Data High register D08 Location Monitor **SHALL** detect odd byte accesses only.

Data Low Register : A write to this register is the least significant word of write data. A write to this register causes the device to execute some action based on the data written to the Data High, Data Low and Data Extended registers. A read of this register is the least significant word of read data.

RULE C.2.57:
There **SHALL** be a Read mode and a Write mode Location Monitor at the Data Low register's address.

RULE C.2.58:
IF a Message Based Device supports D08(EO) accesses of its Data Low register,
THEN its Data Low register D08 Location Monitors **SHALL** detect odd byte accesses only.

OBSERVATION C.2.70:
The Location Monitors at the data registers provide the device with the ability to detect the width of the data or command being sent. 16 bit, 32 bit and 48 bit data transfers can all be independently detected by this information.

A24 Pointer Register: This 32 bit register is defined by the optional Shared Memory protocol. See Section G, "Shared Memory Protocols".

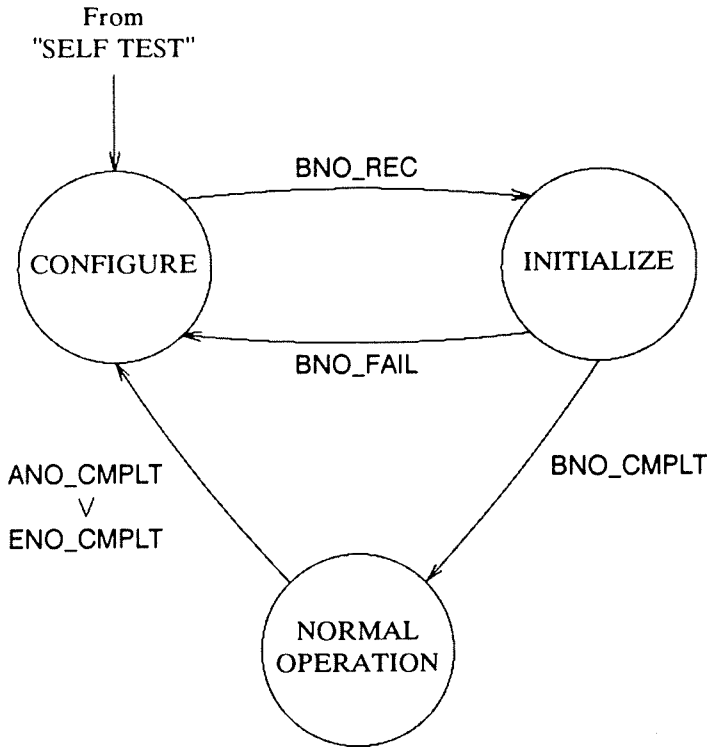
A32 Pointer Register: This 32 bit register is defined by the optional Shared Memory protocol. See Section G, "Shared Memory Protocols".

VXIbus Reserved Registers: These registers are reserved for future VXIbus definition. The effects of reads and writes of these registers are undefined.

C.2.4.4 MESSAGE BASED DEVICE OPERATION

The specification of standard communication protocols requires that all Message Based Devices have a uniform and clearly defined behavior when using common resources. For a Message Based Device, three sub-states are added in the PASSED state defined in Section C.2.1.2, "Device Initialization and Diagnostics". The first, CONFIGURE sub-state, is the state in which a Message Based Device is configured. The second, NORMAL OPERATION sub-state, is the state which is entered when the configuration has been completed. Transitions between these states are initialized by the *Begin Normal Operation, End Normal Operation*, and *Abort Normal Operation* commands. The third sub-state, INITIALIZE, is the state of the device while it is executing the *Begin Normal Operation* command.

The sub-states and the transitions between them are described in Figure C.5 as an extension to Figure C.4. All state transitions already defined in Figure C.4 (Section C.2.1.2, "Device Initialization and Diagnostics") are the same.



BNO_FAIL: *Begin Normal Operation* command failed
BNO_REC: *Begin Normal Operation* command received
BNO_CMPLT: *Begin Normal Operation* command successfully completed
ENO_CMPLT: *End Normal Operation* command successfully completed
ANO_CMPLT: *Abort Normal Operation* command successfully completed

Figure C.5. Configure State Diagram

OBSERVATION C.2.71:
The CONFIGURE, INITIALIZE, and NORMAL OPERATION sub-states are undefined in all states other than PASSED.

RULE C.2.59:
All Message Based Devices **SHALL** support word serial transfers when in the PASSED state (See Section C.2.1.2, "Device Initialization and Diagnostics").

OBSERVATION C.2.72:
Some Message Based Devices may exhibit very long startup, or boot, times. Such devices may be unable to receive or respond to word serial commands for a significant time interval following power-on. Such devices will hold off word serial transactions by the appropriate Response register indications, i.e. by holding the *Write Ready* and/or *Read Ready* bits to zero (0). There is no maximum response time requirement for such devices, and they will delay the system configuration process executed by the Resource Manager. When a Resource Manager encounters such a device it may either wait indefinitely for a response, or time out within some reasonable period.

RULE C.2.60:
While in the CONFIGURE sub-state, a Message Based Device **SHALL** remain deactivated.

OBSERVATION C.2.73:
A deactivated device may not assert any VMEbus IRQ lines or initiate any data transfer cycles.

C.2.4.4.1 State Dependency of Commands

A Message Based Device responds to different sets of commands depending on the sub-state it is in: CONFIGURE, INITIALIZE, or NORMAL OPERATION. Some commands are used in all states. Whether the commands in the sets are mandatory or optional is defined in Section C.2.4.4.2, "Required Commands".

The commands responded to in the CONFIGURE sub-state are called configuration commands:

- *Assign Handler Line*
- *Assign Interrupter line*
- *Grant Device*
- *Identify Commander*
- *Release Device*
- *Read Servant Area*

The commands responded to in the NORMAL OPERATION sub-state are called normal operation commands:

- *Byte Available*
- *Byte Request*
- *Clear Lock*
- *Read STB*
- *Set Lock*
- *Trigger*
- *User Defined*

The commands responded to in both the CONFIGURE and the NORMAL OPERATION sub-states are called state independent commands:

- *Abort Normal Operation*
- *Asynchronous Mode Control*
- *Begin Normal Operation*
- *Clear*
- *Control Event*
- *Control Response*
- *End Normal Operation*
- *Read Handler Line*
- *Read Handlers*

- *Read Interrupter Line*
- *Read Interrupters*
- *Read MODID*
- *Read Protocol*
- *Read Protocol Error*
- *Set Lower MODID*
- *Set Upper MODID*

RULE C.2.61:
A Commander **SHALL NOT** send any normal operation commands to a Servant which is in the CONFIGURE sub-state.

RULE C.2.62:
A Commander **SHALL NOT** send any configure commands to a Servant which is in the NORMAL OPERATION sub-state.

OBSERVATION C.2.74:
It is a protocol violation to send any commands to any device while it is in the INITIALIZE sub-state.

C.2.4.4.2 Required Commands

This section defines minimal conformance requirements for Message Based Devices in terms of their capabilities and sub-states. In this context a phrase of the form "A Message Based Device shall properly respond to..." means that:

- The Message Based Device must support at least the minimum capability implied by the command.
- The Message Based Device must recognize and interpret the command in a manner consistent with the intent of the command.
- The Message Based Device must implement full functional support of the command, according to its capabilities.
- If the command requires a response, the Message Based Device's response must conform to the response formats defined in Section E, "Command and Event Formats".

RULE C.2.63:
All Message Based Devices **SHALL** properly respond to the following commands in both the CONFIGURE and NORMAL OPERATION sub-states:

- *Abort Normal Operation*
- *Begin Normal Operation*
- *Clear*
- *End Normal Operation*
- *Read Protocol*
- *Read Protocol Error*

RULE C.2.64:
All Message Based Devices with VMEbus Master capability **SHALL** properly respond to the following command in the CONFIGURE sub-state:

- *Identify Commander*

OBSERVATION C.2.75:
VMEbus Master capability is indicated by the *Master** bit in the Protocol register.

RULE C.2.65:
All Message Based Commanders **SHALL** properly respond to the following commands in the CONFIGURE sub-state:

- *Read Servant Area*
- *Grant Device*
- *Release Device*

OBSERVATION C.2.76:
Message Based Commander capability is indicated by the *CMDR** bit in the Protocol register.

RULE C.2.66:
All Message Based Devices having Programmable Handler capability **SHALL** properly respond to the following commands in both the CONFIGURE and NORMAL OPERATION sub-states:

- *Read Handlers*
- *Read Handler Line*

RULE C.2.67:
All Message Based Devices having Programmable Handler capability **SHALL** properly respond to the following command in the CONFIGURE sub-state:

- *Assign Handler Line*

OBSERVATION C.2.77:
Programmable Handler capability is indicated by the response to the *Read Protocol* command.

RULE C.2.68:
All Message Based Devices having Programmable Interrupter capability **SHALL** properly respond to the following commands in both the CONFIGURE and NORMAL OPERATION sub-states:

- *Read Interrupters*
- *Read Interrupter Line*

RULE C.2.69:
All Message Based Devices having Programmable Interrupter capability **SHALL** properly respond to the following command in the CONFIGURE sub-state:

- *Assign Interrupter Line*

OBSERVATION C.2.78:
Programmable Interrupter capability is indicated by the response to the *Read Protocol* command.

RULE C.2.70:
IF a Message Based Servant implements response interrupts or response signals,
THEN it **SHALL** have the Response Generation (RG) capability.

RULE C.2.71:
All Message Based Devices having Response Generation capability **SHALL** properly respond to the following commands in the CONFIGURE or NORMAL OPERATION sub-state:

- *Asynchronous Mode Control*
- *Control Response*

RULE C.2.72:
A Message Based Device having Response Generation capability **SHALL** enable or disable response generation on each Response register bit transition for which it supports response generation, in response to the *Control Response* command.

OBSERVATION C.2.79:
Response Generation capability can be determined by the Read Protocol command.

RULE C.2.73:
IF a Message Based Servant implements event interrupts or event signals,
THEN it **SHALL** have the Event Generation (EG) capability.

RULE C.2.74:
All Message Based Devices having Event Generation capability **SHALL** properly respond to the following command in the CONFIGURE or NORMAL OPERATION sub-state:

- *Asynchronous Mode Control*
- *Control Event*

RULE C.2.75:
A Message Based Device having Event Generation capability **SHALL** enable or disable event generation for each event it is capable of generating, in response to the *Control Event* command.

OBSERVATION C.2.80:
Event Generation capability can be determined by the Read Protocol command.

C.2.4.5 MESSAGE BASED DEVICE CONFIGURATION

The specification of standard communication protocols requires that all Message Based Devices be configured in a uniform fashion after a Hard Reset, Soft Reset, or after receiving one of the word serial *Abort Normal Operation, End Normal Operation* or *Clear* commands.

C.2.4.5.1 The Default Configuration

Message Based Devices have a uniform default configuration. The default configuration is assumed when the devices are reset or receive the *Abort Normal Operation* command. The configuration may be modified (from the default) with the configuration commands. The resulting configuration becomes active when the device enters the NORMAL OPERATION sub-state.

RULE C.2.76:
Upon entering the PASSED state, a Message Based Device **SHALL** assume the CONFIGURE sub-state with the default configuration wherein:

1. **IF** the Message Based Device is a Commander,
   THEN its Servant list is empty.
2. Responses and events are globally enabled (See the definition of the *Asynchronous Mode Control* command in Section E.1, "Word Serial Commands").

3. **IF** the Message Based Device has VMEbus Master capability,
   THEN responses and events are sent via signals to Logical Address 0 (zero).
4. **IF** the Message Based Device does not have VMEbus Master capability,
   THEN responses and events are sent via interrupts.
5. **IF** the Message Based Device has Programmable Interrupter capability,
   THEN the Interrupter capabilities are not connected to any IRQ lines.
6. **IF** the Message Based Device has Programmable Handler capability,
   THEN the Interrupt Handler capabilities are not connected to any IRQ lines.
7. Generation of all supported events is enabled. (See the definition of the *Control Event* command in Section E.1, "Word Serial Commands".)
8. Generation of B14*, DOR*, DIR*, Err*, RR*, WR*, FHS* responses is disabled. (See the definition of the *Control Response* command in Section E.1, "Word Serial Commands".)
9. The device is not processing any incoming signals or interrupts.

OBSERVATION C.2.81:

Since the device is de-activated in the CONFIGURE sub-state, even though responses and events may be enabled, signals and interrupts are not generated until the device enters the NORMAL OPERATION sub-state.

The default configuration is summarized in Table C.1.

DEFAULT CONFIGURATION	
Identify Commander:	Resource Manager
Asynchronous Mode:	
Events	Enabled (via signals if Bus Master)
Responses	Enabled (via signals if Bus Master)
Control Response:	All Bits Disabled
Control Event:	All Events Enabled
Programmable Handlers	All Disconnected
Programmable Interrupters	All Disconnected
Servant List	Empty

TABLE C.1. The Default Configuration.

C.2.4.5.2 Modifying The Configuration

Changes can be made to the current configuration in the CONFIGURE sub-state using the configuration commands and state independent commands.

RULE C.2.77:
Upon receipt of the *Grant Device* command, a Commander **SHALL** add the granted device to its Servant list.

OBSERVATION C.2.82:
Each Commander maintains a list of the Logical Addresses of Servant devices, referred to as the Servant list.

RULE C.2.78:
Upon receipt of the *Release Device* command, a Commander **SHALL** delete the released device from its Servant list.

RULE C.2.79:
IF a Message Based Device supports the *Identify Commander* command,
THEN upon receipt of the *Identify Commander* command, the Message Based Device **SHALL** register the identified Commander as the device's Commander.

RULE C.2.80:
IF a Message Based Device supports the *Assign Handler Line* command,
THEN upon receipt of the *Assign Handler Line* command the Message Based Device **SHALL** connect the IRQ line to the Interrupt Handler capability.

RULE C.2.81:
IF a Message Based Device supports the *Assign Interrupter Line* command,
THEN upon receipt of the *Assign Interrupter Line* command the Message Based Device **SHALL** connect the IRQ line to the Interrupter capability.

C.2.4.6 INITIATING OPERATION

Once configured, a Message Based Device is activated via the *Begin Normal Operation* command. Receipt of the *Begin Normal Operation* command causes a device to initialize its Servants and enter the NORMAL OPERATION sub-state. While in the NORMAL OPERATION sub-state, the device will have its *Ready* bit set to one (1).

OBSERVATION C.2.83:
While in the NORMAL OPERATION sub-state the Commander/Servant hierarchy and the interrupt line configuration cannot be changed, according to Section C.2.4.4.1, "State Dependency of Commands".

OBSERVATION C.2.84:
The *Begin Normal Operation* command indicates, via the *Top Level* bit, whether a device is a Top Level device or not. A top level device has no Commander, and in the NORMAL OPERATION sub-state it may operate autonomously with respect to all other devices except a device involved in runtime resource management. A device involved in runtime resource management retains the capability of forcing a top level device into the CONFIGURE sub-state by sending the *End Normal Operation* or the *Abort Normal Operation* command to the top level device.

RULE C.2.82:
Upon the receipt of a *Begin Normal Operation* command, a Commander which is in the CONFIGURE sub-state **SHALL** execute the following process.

1. Enter the INITIALIZE sub-state.
2. Discard all pending and initiated signals and VMEbus interrupts.
3. Determine which Message Based Servants are VMEbus Masters by reading the Protocol register.
4. Send the *Identify Commander* command to all Message Based Servants which are VMEbus Masters.
5. Send the *Begin Normal Operation* command to all Message Based Servants and wait for their responses.

6. Configure all its non-Message Based Servants as needed. This includes allocating IRQ lines.
7. If the above process is successful, enter the NORMAL OPERATION sub-state and set to one (1) the *Ready* bit of the Status register. Otherwise, return to the CONFIGURE sub-state.
8. Return the appropriate response, indicating the state(s) of itself and its hierarchical Servants, in the Data Low register. This response will incorporate any device failure information returned (in response to *Begin Normal Operation*) by the Commander's Servants.

OBSERVATION C.2.85:
In general, a device will enter the NORMAL OPERATION sub-state if it is capable of normal operation.

RULE C.2.83:
Upon the receipt of a *Begin Normal Operation* command, a Commander which is in the NORMAL OPERATION sub-state **SHALL** return the appropriate response, indicating the state(s) of itself and its hierarchical Servants, in the Data Low register.

OBSERVATION C.2.86:
Since a device in the NORMAL OPERATION sub-state has already initialized its Servants, no action other than the response specified in the previous rule is required.

RULE C.2.84:
A Message Based Device which is in either the CONFIGURE or INITIALIZE state **SHALL** have the *Ready* bit of its Status register cleared to zero (0).

RULE C.2.85:
A Message Based Device which is in the NORMAL OPERATION sub-state **SHALL** have the *Ready* bit of its Status register set to one (1).

OBSERVATION C.2.87:
Normally, the Resource Manager will configure a Message Based Device with PI capability to drive an IRQ line that its Commander handles.

OBSERVATION C.2.88:
A Servant may be programmed to drive an IRQ line which is not handled by its Commander.

OBSERVATION C.2.89:
It is the responsibility of the system integrator to ensure that the IRQ lines are properly allocated.

RULE C.2.86:
A VXIbus device **SHALL NOT** attempt to control, through the VXIbus registers, any VXIbus device not in its own Servant list.

RULE C.2.87:
A Commander lacking device specific knowledge **SHALL NOT** access any of its Servants' device dependent registers other than the VXIbus defined registers.

OBSERVATION C.2.90:
It is not safe to assume that a Commander may freely write to any device dependent register of a Servant Device. Such accesses should not (in general) be attempted without good cause.

C.2.4.7 TERMINATING OPERATION

Four methods are available for terminating the operation of Message Based Devices. These are Hard Reset, Soft Reset, *Abort Normal Operation* and *End Normal Operation*. Each of these differ in the level of reconfiguration they impose on the device. Table C.2 outlines the actions imposed by these methods. Also included in the table are the actions imposed by the *Clear* command. Though the *Clear* command, as defined for Message Based Devices (Section C.2.4.8, "Clearing a Message Based Device"), does not terminate the operation of the received device, it has been included in this table for completeness.

	Hard Reset	**Soft Reset**	**ANO**	**ENO**	**Clear**
Generated By	SYSRESET*	Reset Bit	*Abort Normal Operation* Command	*End Normal Operation* Command	*Clear* Command
Action Defined	Upon Entry to PASSED State	Upon Entry to PASSED State	After Command Processed	After Command Processed	After Command Processed
All Devices:					
Sub-State	CONFIGURE	CONFIGURE	CONFIGURE	CONFIGURE	Unchanged
Configuration	Default	Default	Default	Unchanged	Unchanged
Status Register:					
A24/A32 Active	0	Unchanged	Unchanged	Unchanged	Unchanged
Offset Register:	Initial	Unchanged	Unchanged	Unchanged	Unchanged
Logical Address:	Initial	Unchanged	Unchanged	Unchanged	Unchanged
Response Register:					
*Err** bit	1	1	1	1	1
Read Ready	0	0	Valid	Valid	0
*FHS Active**	1	1	1	1	Unchanged
*Locked**	1	1	1	1	Unchanged
Bus Masters:					
Data Transfers	No	No	No	No	Unchanged
Interrupters:					
IRQ Assertions	No	No	No	No	Unchanged

TABLE C.2. Message Based Device Termination and Clear Actions

C.2.4.7.1 Hard Reset

Upon receipt of a Hard Reset (assertion of SYSRESET*), a Message Based Device enters, as does any other device type, the HARD RESET state. From that state a device continues its initialization until it ends in one of the possible states as defined in Section C.2.1.2, "Device Initialization and Diagnostics". A Message Based Device will, if it enters the PASSED state, enter the CONFIGURE sub-state configured with the default configuration.

Hard Reset is used only when it is necessary to promptly terminate operation of all devices in a VXIbus system. A Hard Reset always affects the entire VXIbus system, and will result in the system being configured by the Resource Manager.

C.2.4.7.2 Soft Reset

Upon receipt of a Soft Reset (setting and clearing the *Reset* bit), a Message Based Device enters, as does any other device type, the SOFT RESET state. From that state a device passes through SELF TEST state until it ends in one of the possible states as defined in Section C.2.1.2 "Device Initialization and Diagnostics". A Message Based Device will, if it enters the PASSED state, enter the CONFIGURE sub-state configured with the default configuration.

A Soft Reset affects only a single device and is used to deactivate a device. A deactivated device is prevented from using the VXIbus.

RECOMMENDATION C.2.19:
In general, a Commander should Soft Reset a Message Based Servant only if all VXIbus termination commands fail.

OBSERVATION C.2.91:
A Message Based Device that has been Soft Reset will have all of its configuration parameters reset (to the Default Configuration). Its state may also be inconsistent with the remainder of the system, due to lost signals or abandoned communication links with other VXIbus devices. It is important that such a device not be activated until its configuration is restored and all inconsistencies have been resolved. In general, the Resource Manager is the only entity capable of doing so.

RULE C.2.88:
When a Commander soft resets a device it **SHALL** ensure that operation of all Servants of that device is also terminated.

OBSERVATION C.2.92:
In general, the Resource Manager is the only device with full knowledge of Servant hierarchies.

RULE C.2.89:
Before a Commander sends *Begin Normal Operation* to a Servant after having soft reset that Servant, the Commander **SHALL** guarantee that the device and the system are in a consistent configuration and state.

OBSERVATION C.2.93:
When a Commander receives *Begin Normal Operation* while it is in the CONFIGURE sub-state, the Commander can assume that all its Servants are in a state and configuration consistent with the system.

C.2.4.7.3 Abort Normal Operation

An *Abort Normal Operation* affects only a single device and is used by a Commander or by the Resource Manager to deactivate a device i.e. prevent it from using the VXIbus. After an *Abort Normal Operation*, a Message Based Device will enter the CONFIGURE sub-state configured with the default configuration. In contrast to the Soft Reset, the device does not pass through the SELF TEST state.

RECOMMENDATION C.2.20:
In general, a Commander should send the *Abort Normal Operation* command to one of its Servants only after an attempt to send *End Normal Operation* had already failed.

RULE C.2.90:
Upon the receipt of the *Abort Normal Operation* command in the NORMAL OPERATION sub-state, a Message Based Device **SHALL** respond in the following manner:

1. Abort operation as fast as possible.
2. Exit the NORMAL OPERATION sub-state to the CONFIGURE sub-state, clearing the *Ready* bit of its Status register to zero (0).
3. Stop processing incoming signals and interrupts.
4. Restore the default configuration (the Logical Address and programmed offset is kept).
5. Return the appropriate code in the Data Low register.

RULE C.2.91:
Upon the receipt of the *Abort Normal Operation* command in the CONFIGURE sub-state, a Message Based Device **SHALL** respond in the following manner:

1. Restore the default configuration (the Logical Address and programmed offset is kept).
2. Return the appropriate code in the Data Low register.

OBSERVATION C.2.94:
As a consequence of (re-) entering the CONFIGURE sub-state, the response signal or interrupt generation is disabled and all pending bus requests and interrupts are unasserted.

OBSERVATION C.2.95:
Abort Normal Operation is not propagated through the hierarchy.

OBSERVATION C.2.96:
A device that has been Aborted will have all of its configuration parameters reset (to the Default Configuration). Its state may also be inconsistent with the remainder of the system, due to lost signals or abandoned communication links with other VXIbus devices. It is important that such a device not be activated until its configuration is restored and all inconsistencies have been resolved. In general, the Resource Manager is the only entity capable of doing so.

RULE C.2.92:
When a Commander terminates the operation of a device with the *Abort Normal Operation* command it **SHALL** ensure that operation of all Servants of that device is also terminated.

OBSERVATION C.2.97:
In general, the Resource Manager is the only device with full knowledge of Servant hierarchies.

RULE C.2.93:
Before a Commander sends *Begin Normal Operation* to a Servant after having aborted that Servant's operation, the Commander **SHALL** guarantee that the device and the system are in a consistent configuration and state.

OBSERVATION C.2.98:
When a Commander receives *Begin Normal Operation* while it is in the CONFIGURE sub-state, the Commander can assume that all its Servants are in a state and configuration consistent with the system.

C.2.4.7.4 End Normal Operation

An *End Normal Operation* command affects a hierarchy of devices and is used by a Commander or by the Resource Manager to bring the devices into a state from which they can be restarted. After a successful *End Normal Operation* command, the devices in the hierarchy are deactivated.

RULE C.2.94:
Upon the receipt of the *End Normal Operation* command in the NORMAL OPERATION sub-state, a Message Based Device **SHALL** respond in the following manner:

1. Allow operation to complete in a device specific consistent manner. All operation shall eventually end, but there is no time limit.
2. Bring any Register Based or Extended Device Servants into a deactivated state.
3. Send the *End Normal Operation* command to all its Message Based Servants and wait for their responses.
4. Stop processing incoming signals and interrupts.
5. Exit the NORMAL OPERATION sub-state to the CONFIGURE sub-state, clearing the *Ready* bit of the Status register to zero (0). The current configuration is kept.
6. Return the appropriate response code in the Data Low register. This response will incorporate any device failure information returned (in response to *End Normal Operation*) by the Commander's Servants.

RULE C.2.95:
Upon the receipt of the *End Normal Operation* command in the CONFIGURE sub-state, a Message Based Device **SHALL** return the appropriate response code in the Data Low register.

OBSERVATION C.2.99:
Since a device in the CONFIGURE sub-state has already been deactivated, no action other than the response specified in the previous rule is required.

OBSERVATION C.2.100:
As a consequence of entering the CONFIGURE sub-state, the device is deactivated i.e. prevented from using the VXIbus.

OBSERVATION C.2.101:
Those state parts (e.g. higher level configurations) which are not covered by the rules in this section can upon reception of End Normal Operation be changed in any manner that is device specific or may be standardized in the future.

OBSERVATION C.2.102:
The *End Normal Operation* command is propagated from Commander to Servant until it has reached the bottom of the hierarchy.

OBSERVATION C.2.103:
It is device-dependent if or how a device resumes operation on receipt of *Begin Normal Operation* after *End Normal Operation*.

C.2.4.8 CLEARING A MESSAGE BASED DEVICE

A *Clear* command is used only to remove pending output data from a Message Based Device's Data Low, High and Extended registers. Just as *Abort Normal Operation* and *End Normal Operation*, it may have an additional meaning to higher level protocols.

RULE C.2.96:
Upon the receipt of the *Clear* command, a Message Based Device **SHALL**:

- Remove any data output in the Data Low, Data High and Data Extended registers.
- Remove any buffered data output to the Data Low, Data High and Data Extended registers.
- Discard output data from any output data generating operations previously started.
- Become ready to receive a new command.
- Clear (to 0) the *Read Ready* bit of its Response register.
- De-assert the *Err** bit of the Response register and cancel any pending *Read Protocol Error* command responses.

RULE C.2.97:
The *Clear* command **SHALL NOT** affect:

- The CONFIGURE/NORMAL OPERATION sub-state.
- The PI or PH settings.
- The Servant list.
- The Signal register.
- Assertion of IRQ lines.
- Communication links with other VXIbus devices.

OBSERVATION C.2.104:
The effects of *Clear* on a VXIbus instrument are documented in Section D.1.1.3, "Clearing a VXIbus Instrument".

C.2.5 Extended Devices

An Extended device is a VXIbus device with provisions for defining new subclasses of VXIbus devices for future applications. Each Extended Device has a subclass register in addition to its configuration registers. The subclass register allows for the definition of both standard and manufacturer specific Extended Device subclasses.

OBSERVATION C.2.105:

An Extended Device implements the standard set of VXIbus configuration registers.

RULE C.2.98:

An Extended Device **SHALL** implement a subclass register.

The Extended Device register map is illustrated in the following table:

$3F_{16}$	SUBCLASS DEPENDENT REGISTERS
20_{16}	
$1E_{16}$	Subclass Register
$1D_{16}$	SUBCLASS DEPENDENT REGISTERS
08_{16}	
00_{16}	CONFIGURATION REGISTERS

The registers are defined as follows:

Subclass Register: The read contents of this 16 bit register indicate the subclass of the device. This register has two alternate formats depending on the value of bit 15:

Bit #	15	14 ← 0
Contents	1	Reserved Subclass

Bit #	15	14 ← 12	11 ← 0
Contents	0	Manufacturer Subclass	Manufacturer ID

Bit 15 is used to differentiate between VXIbus defined subclasses and manufacturer defined subclasses. The other fields are defined as follows:

- Reserved Subclass: These bits identify the device's subclass. All values of this field are reserved for future VXIbus definition.
- Manufacturer Subclass: These bits indicate the device's subclass as defined by a particular VXIbus manufacturer. The values of this field are defined by the manufacturer indicated by the adjacent *Manufacturer ID* field.
- Manufacturer ID: These bits identify the manufacturer responsible for the definition of this particular subclass. The format is the same as in the *Manufacturer ID* field of the ID register (See Section C.2.1.1.2, "Configuration Registers").

OBSERVATION C.2.106:
Each VXIbus manufacturer may define up to 8 Extended Device Subclasses. Each of these subclasses has a unique Manufacturer Subclass/ID combination.

The effects of writes to the Subclass Register location are subclass dependent.

Subclass Dependent Registers: These registers will be defined according to a device's subclass requirements as a part of that subclass's definition.

C.3 DEVICE COMMUNICATION PROTOCOLS:

This section describes the protocols necessary for proper communication between VXIbus devices.

C.3.1 Communication Elements

C.3.1.1 REGISTER BASED SERVANTS

Register Based Servant describes the communication element of a Register Based device. A Register Based Servant responds to no defined protocols, since the device's registers are entirely card dependent.

C.3.1.2 MESSAGE BASED SERVANTS

These devices are generally capable of independently executing complex commands, and may also control other devices in a hierarchical instrumentation system. Message Based Servants communicate using the VXIbus Message Based device protocols.

C.3.1.3 MESSAGE BASED COMMANDERS

A Message Based Commander is the interface for a Message Based Device exercising control over other devices. Message Based Commanders communicate using the VXIbus Message Based Device protocols.

C.3.2 Register Based Servant Control

The protocols for control of Register Based Devices are completely device dependent. The designer of a Register Based Device is free to specify any register interactions and control protocols required for proper operation of that device.

C.3.3 Message Based Servant Control

The protocols for Commander → Servant communication involve the Servant's Protocol, Response, and Data registers. They may optionally utilize the Commander's Signal register or the VMEbus interrupts. The simplest communications use the data and Response registers to transfer data in a word serial fashion. This mode is defined as the **Word Serial** protocol. All Message Based Devices implement 16-bit word serial transfers. The 16-bit word serial transfer is the most basic method of communication defined for Message Based Devices. It is simple to implement (in both hardware and software), yet provides the required communication capability to achieve system objectives.

Communication between Message Based Devices is established with 16-bit word serial transfers, then, upon notification of device capabilities, may progress to more sophisticated protocols. These may include higher performance word serial transfers as well as shared memory protocols. A Commander uses the *read protocol* command to determine which higher performance protocols are supported by a Servant.

RULE C.3.1:
All Message Based Devices **SHALL** implement the *read protocol* command.

C.3.3.1 WORD SERIAL PROTOCOLS

The **Word Serial Protocol** is a communication protocol in which data words are transferred serially to or from a fixed location, in this case the read or write data register of the Servant.

The word serial protocols are based upon a generalized model of a full duplex UART (Universal Asynchronous Receiver/Transmitter). Each implementation utilizes bidirectional data registers and a Response register. The data registers are categorized as full duplex, because reads and writes are totally independent. Each write to the data registers is interpreted as a command, unless a previous command has defined it as data. Commands may contain embedded data or may require supporting data to be transferred in succeeding writes. Such command/data sequences are generally not interruptible.

Progress of a data transfer is paced by bits in the Response register which indicate whether the write data registers are empty and whether the read data registers are full.

Data may be written into the write data registers only when the *Write Ready* bit in the Response register is set (1). When data is placed in the write data registers, the *Write Ready* bit is cleared (to 0) until the data is accepted by the Servant.

Valid data may be read from the read data registers only when the *Read Ready* bit in the Response register is set (1). When the data is read from the read data registers, the *Read Ready* bit is cleared (to 0) until the Servant places another word in the read data registers.

A Servant will source data to its read data registers only in response to a command. It is up to its Commander to read such data before issuing a command requiring the output of more data. Thus the Servant is not required to maintain an output queuing structure.

PERMISSION C.3.1:
A Commander **MAY** issue a command requiring a Servant to write more than one response to its read data registers.

RULE C.3.2:
A Servant **SHALL NOT** source any data to its read data registers except in response to an explicit request (for one or more data words) by that device's Commander.

OBSERVATION C.3.1:
If a Commander reads a Servant's data registers without first explicitly requesting the data, the Servant will either respond by causing a bus error or return unspecified data (with or without an error indication). Thus the Servant will either prevent the erroneous read, detect the error, or ignore the protocol violation.

RULE C.3.3:
A Commander **SHALL NOT** send any command requiring a Servant to place data in its data registers until the Commander has read (from the data registers) all data generated by previous commands (and the *Read Ready* bit is zero (0)).

OBSERVATION C.3.2:
The preceding rule does NOT restrict the implementation of output queuing structures within higher level protocols.

Three forms of word serial communications are defined in this section: Word Serial transfers (16 bit), Longword Serial transfers (32 bit), and Extended Longword Serial transfers (48 bit). Support for the Longword and Extended Longword serial transfers is optional. Data transfers utilizing any of these three protocols may be freely intermixed with data transfers of the other two protocols.

OBSERVATION C.3.3:
Longword and Extended Longword transfers to a Servant that does not support these protocols is meaningless, and will have unpredictable results.

OBSERVATION C.3.4:
Every Message Based Device has write mode Location Monitors at each of its supported data registers (Data Low, Data High, and Data Extended). These Location Monitors indicate to the device which protocol is being invoked by its Commander for each transfer.

OBSERVATION C.3.5:
The command sets for all forms of word serial communication are orthogonal (mutually exclusive). The command sets are described in Section E, "Command and Event Formats".

C.3.3.1.1 Word Serial Transfers

The word serial transfer is the minimum usable data transfer protocol. The data path width is 16 bits (one word).

Data is transferred by reads or writes of the Data Low register. By default, all writes are interpreted as commands. Each transfer affects the state of the appropriate bit (*Read Ready* or *Write Ready*) of the Response register.

RULE C.3.4:
IF a word serial data transfer requires more than a single bus cycle,
THEN the least significant byte must be transferred last, to ensure validity of the associated *Read Ready* or *Write Ready* bit.

RULE C.3.5:
All Message Based Devices **SHALL** always respond to the Word Serial protocol.

C.3.3.1.2 Longword Serial Transfers

The longword serial transfer is a higher performance protocol than the word serial, having a 32 bit wide data path.

Data is transferred by reads or writes of the Data High and Data Low registers. By default, all writes are interpreted as commands. Each Longword Serial transfer affects the state of the appropriate bit (*Read Ready* or *Write Ready*) of the Response register.

RULE C.3.6:
IF a longword serial data transfer requires more than a single bus cycle,
THEN the least significant word or byte must be transferred last, to ensure validity of the associated *Read Ready* or *Write Ready* bit.

OBSERVATION C.3.6:
Support of the Longword Serial communication protocol is optional for Message Based Devices. This capability is indicated in the response to the *read protocol* command (see Section E.1, "Word Serial Commands").

C.3.3.1.3 Extended Longword Serial Transfers

The extended longword serial transfer is a higher performance protocol than the longword serial, having a 48 bit wide data path. This is a write only protocol.

Data is transferred by writes to the Data Extended, Data High, and Data Low registers. By default, all writes are interpreted as commands. Each Extended Longword Serial transfer affects the state of the *Write Ready* bit of the Response register.

RULE C.3.7:
IF an extended longword serial data transfer requires more than a single bus cycle,
THEN the least significant word or byte must be transferred last, to ensure validity of the associated *Read Ready* or *Write Ready* bit.

PERMISSION C.3.2:
Support of the Extended Longword Serial communication protocol is optional for Message Based Devices. This capability is indicated in the response to the *read protocol* command (see Section E.1, "Word Serial Commands").

C.3.3.2 FAST HANDSHAKE TRANSFERS

The word serial protocols may utilize either of two handshake modes for data transfer, the **Normal Transfer** mode or the **Fast Handshake** mode. In the normal transfer mode, a Commander uses bits in the Servant's Response register to synchronize data transfers. In the Fast Handshake mode, a Commander is allowed to execute word serial data transfers without checking the Servant's Response register.

The Fast Handshake mode uses the Servant's DTACK* and BERR* lines to enforce proper synchronization. In this mode, a Servant may hold off a VMEbus transfer for up to 20 μs until it is ready to complete the transaction. If the 20 μs expires before the appropriate readiness condition is true, the Servant will assert BERR*. Otherwise the Servant will assert DTACK* and complete the transfer normally.

When in the Fast Handshake mode, a Servant is always ready to accept certain defined word serial operations. These operations are defined by higher-layer protocols such as the Byte Transfer protocol (See Section C.3.3.3, "Byte Transfer Protocol"). In the Fast Handshake mode a Commander is allowed to send certain standard word serial commands which may be transmitted using either Normal Handshake or Fast Handshake methods. These are referred to as **FHS Commands**.

RULE C.3.8:
A Commander **SHALL NOT** attempt any word serial operations using the Fast Handshake protocol, unless such operations are executed within the context of a higher-layer protocol using only the FHS commands defined for that protocol.

OBSERVATION C.3.7:
Presently, the only defined FHS commands are the *Byte Available* and *Byte Request* commands used in the Byte Transfer Protocol. Future VXIbus specifications may define additional FHS commands.

C.3.3.2.1 Servant Fast Handshake Operation

RULE C.3.9:
A Message Based Servant **SHALL** always support the normal transfer mode, even when in the Fast Handshake mode.

OBSERVATION C.3.8:
A FHS capable Servant supports the normal transfer mode by always indicating its true state via the *Write Ready, Read Ready, DIR, DOR,* and *Err** bits in its Response register.

PERMISSION C.3.3:
When in the Fast Handshake mode, a Servant **MAY** set any of its *Write Ready, Read Ready, DIR,* or *DOR* bits in its Response register to one (1), up to 20μs before the indicated condition becomes true.

RECOMMENDATION C.3.1:
When in Fast Handshake mode, a Servant should control its Response register bits in a manner that allows normal mode transfers to execute without long VMEbus data transfer cycles.

A Servant indicates its support of the Fast Handshake mode via the *FHS** bit in its protocol register. The *FHS Active** bit of its Response register indicates the current status of the Servant's Fast Handshake mode. The meanings of these bits are summarized in the following table.

BIT	Value	
	0	1
FHS*	Capable	Incapable
FHS Active*	Active	Inactive

PERMISSION C.3.4:
A Servant **MAY** initiate the Fast Handshake mode by clearing the *FHS Active** bit, at a time defined by the applicable higher-layer protocol.

Because a Commander in the Fast Handshake mode does not test the *Read Ready* and *Write Ready* bits, a FHS capable Servant is required to prevent incorrect data transfers by asserting BERR*. Such incorrect transfers may be caused by Read Ready protocol violations, Write Ready protocol violations, higher-layer protocol violations, or the Servant's inability to keep up with the Commander's data flow rate, either before or after the Servant has exited the FHS active mode.

RULE C.3.10:
A Fast Handshake capable Servant **SHALL** assert BERR* if it can not complete a data transfer cycle accessing its Data Low register within 20 μS.

A Servant may terminate the Fast Handshake mode at any time by setting (to 1) its *FHS Active** bit. It will assert BERR* on the next read or write of the Data Low register that it cannot complete within 20 μS.

A Servant will terminate the Fast Handshake mode any time it asserts BERR* during a data transfer cycle. Under this condition, the Servant updates neither its read nor write queue. It sets (to 1) its *FHS Active** bit, and operates in the normal transfer mode until it is ready to resume the Fast Handshake mode.

C.3.3.2.2 Commander Fast Handshake Operation

The Fast Handshake mode is optional for Commanders, since the normal transfer mode always functions properly. This is true even if its Servant is in the Fast Handshake mode, because its Response register indications are always correct.

A Commander that supports the Fast Handshake mode may execute Fast Handshake transfers only with Servants that are capable of Fast Handshake transfers and which have the Fast Handshake mode active.

RULE C.3.11:
A Commander **SHALL NOT** attempt Fast Handshake mode operations with a Servant having the *FHS** bit of its protocol register set (to one).

RULE C.3.12:
A Commander **SHALL NOT** attempt Fast Handshake operations with a Servant while the Servant's *FHS Active** bit is set (to 1).

A Commander may begin Fast Handshake transfers with a capable Servant only as provided for by the operating higher-layer protocol (provided that the Servant's *FHS Active** bit is asserted (0)). It may exit the fast handshake mode at any time. However, it must exit the Fast Handshake mode after any data transfer attempt that results in a bus error, or at any other time defined by the higher-layer protocol. The Fast Handshake mode must not be resumed until the Servant again clears (to 0) its *FHS* Active* bit.

RULE C.3.13:
IF a word serial data transfer attempt to a Fast Handshake capable Servant results in the assertion of BERR*,
THEN the Commander **SHALL NOT** attempt any Fast Handshake transfers with the same Servant until that Servant clears (to 0) its *FHS Active** bit.

OBSERVATION C.3.9:
No data is transferred during a cycle that causes a bus error. The failed transfer should be retried.

OBSERVATION C.3.10:
Once a Commander detects a Servant in the Fast Handshake active condition, it does not need to check the Servant's *Read Ready*, *Write Ready*, *DIR*, *DOR*, *Err**, or *FHS Active** bits again until the Commander exits the Fast Handshake mode.

C.3.3.3 BYTE TRANSFER PROTOCOL

The **Byte Transfer Protocol** is a mechanism for transferring 8-bit data between Commanders and their Servants. It utilizes the word serial *Byte Available* and *Byte Request* commands. The *Byte Available* command is used to transfer data from a Commander to its Servant. The *Byte Request* command is used to transfer data from a Servant to its Commander. The data flow is regulated by the use of the *DIR* and *DOR* bits in the Servant's Response register.

RULE C.3.14:
When a Message Based Device is not ready to process a *Byte Available* command it **SHALL** clear to zero (0) the *DIR* bit in its Response register.

OBSERVATION C.3.11:
When a Message Based Device is ready to process a *Byte Available* command it will set to one (1) the *DIR* bit in its Response register.

RULE C.3.15:
When a Message Based Device is not ready to process a *Byte Request* command, it **SHALL** clear to zero (0) the *DOR* bit in its Response register.

OBSERVATION C.3.12:
When a Message Based Device is ready to process a *Byte Request* command, it will set to one (1) the *DOR* bit in its Response register.

RULE C.3.16:
IF a word serial command to a Message Based Device causes the de-assertion of the *DIR* bit,
THEN the *Write Ready* bit **SHALL NOT** be set to one (1) until the *DIR* bit is cleared to zero (0).

RULE C.3.17:
IF a word serial command to a Message Based Device causes the de-assertion of the *DOR* bit,
THEN the *Write Ready* bit **SHALL NOT** be set to one (1) until the *DOR* bit is cleared to zero (0).

RULE C.3.18:
A Servant **SHALL NOT** clear (to 0) either its *DIR* or *DOR* bit while its *Write Ready* bit is set (to 1).

RULE C.3.19:
A Commander **SHALL NOT** send a *Byte Available* command to any Servant which has its *DIR* bit cleared to zero (0).

RULE C.3.20:
A Commander **SHALL NOT** send a *Byte Request* command to any Servant which has its *DOR* bit cleared to zero (0).

C.3.3.3.1 Fast Handshake Support

The Byte Transfer Protocol may be executed using the Fast Handshake protocol. Both the *Byte Available* and the *Byte Request* commands are FHS commands, which may utilize the Fast Handshake protocol.

RULE C.3.21:
A Servant that supports the Byte Transfer protocol and the Fast Handshake protocol **SHALL** support Fast Handshake transfers of both the *Byte Available* and *Byte Request* commands.

C.3.3.3.1.1 Byte Available

The FHS protocol may be used to transfer a message using a series of *Byte Available* commands. A Servant capable of receiving *Byte Available* commands in the Fast Handshake mode may enter the FHS Active state (and assert its FHS Active* bit) if it is ready to receive *Byte Available* commands using the Fast Handshake protocol.

A Commander may begin using Fast Handshake transfers to send *Byte Available* commands to a Servant if the Servant's *Write Ready, DIR*, and *ERR** bits are set to one and the *FHS Active** bit is asserted (0).

RULE C.3.22:
A Commander **SHALL NOT** begin Fast Handshake transfers of *Byte Available* commands to a Servant if that Servant's *DIR* bit is zero (0), or its *Write Ready* bit is zero (0), or its *Err** bit is zero (0), or its *FHS Active** bit is one (1).

RULE C.3.23:
A Commander using Fast Handshake to transfer *Byte Available* commands to a Servant **SHALL** exit the Fast Handshake mode before sending any command other than *Byte Available* to that Servant.

OBSERVATION C.3.13:
A Commander is required to exit the Fast Handshake mode when a read or write of the Servant's Data Low register results in a bus error.

The *END* bit of the *Byte Available* command is used to indicate the end of a message. This may cause a servant to exit the Fast Handshake Active state. If the commander also exits the Fast Handshake mode, it can avoid a bus error.

RECOMMENDATION C.3.2:
A Commander should exit the Fast Handshake mode after sending a *Byte Available* command having the *END* bit asserted.

RULE C.3.24:
When its Commander writes a *Byte Available* to a Servant capable of receiving *Byte Available* commands in the Fast Handshake mode, and the Servant's DIR bit is zero, the Servant **SHALL** assert BERR* without changing the state of any of its *Read Ready, Write Ready, DIR, DOR*, or Err* bits.

C.3.3.3.1.2 Byte Request

The FHS protocol may be used to transfer a message using a series of *Byte Request* commands. A Servant capable of receiving *Byte Request* commands in the Fast Handshake mode may enter the FHS Active state (and assert its FHS Active* bit) if it is ready to receive *Byte Request* commands using the Fast Handshake protocol.

A Commander may begin using Fast Handshake transfers to send *Byte Request* commands to a Servant if the Servant's *Write Ready, DOR*, and *ERR** bits are set to one and the *FHS Active** bit is asserted (0).

RULE C.3.25:
A Commander **SHALL NOT** begin Fast Handshake transfers of *Byte Request* commands to a Servant if that Servant's *DOR* bit is zero (0), or its *Write Ready* bit is zero (0), or its *Err** bit is zero (0), or its *FHS Active** bit is one (1).

RULE C.3.26:
A Commander using Fast Handshake to transfer *Byte Request* commands to a Servant **SHALL** exit the Fast Handshake mode before sending any command other than *Byte Request* to that Servant.

OBSERVATION C.3.14:
A Commander is required to exit the Fast Handshake mode when a read or write of the Servant's Data Low register results in a bus error.

The *END* bit of the response to a *Byte Request* command is used to indicate the end of a message. This may cause a servant to exit the Fast Handshake Active state. If the commander also exits the Fast Handshake mode, it can avoid a bus error.

RECOMMENDATION C.3.3:
A Commander should exit the Fast Handshake mode after reading a response to a *Byte Request* command if the *END* bit of the response is asserted.

RULE C.3.27:
When its Commander writes a *Byte Request* to a Servant capable of receiving *Byte Request* commands in the Fast Handshake mode, and the Servant's DOR bit is zero, the Servant **SHALL** assert BERR* without changing the state of any of its *Read Ready, Write Ready, DIR, DOR*, or Err* bits.

C.3.3.4 ERROR HANDLING

VXIbus Message Based Devices implement a uniform method of reporting word serial protocol errors. It is a multi-level approach wherein all word serial protocol conformance errors are reported via the *Err** bit in the Response register and the *Read Protocol Error* command. Higher level errors are reported at higher levels, often as status responses to commands.

There are six defined word serial protocol errors:

- **Unsupported Command:** This error is detected when a Servant receives a command which it does not support.
- **Multiple Query:** This error occurs when a Servant receives a command requiring it to output a response to its Data Low register, and is unable to respond because of an unread response to a previous command.
- **DIR Violation:** This error occurs when a Servant receives a *Byte Available* command when it is unable to process that command because the *DIR* bit is zero (0).
- **DOR Violation:** This error occurs when a Servant receives a *Byte Request* command when it is unable to process that command because the *DOR* bit is zero (0).
- **Write Ready Violation:** This error occurs when data is written to a Servant while its *Write Ready* bit is zero (0).
- **Read Ready Violation:** This error occurs when data is read from a Servant while its *Read Ready* bit is zero (0).

RULE C.3.28:
A Message Based Device **SHALL** detect the *Unsupported Command*, *Multiple Query*, *DIR Violation*, and *DOR Violation* errors.

OBSERVATION C.3.15:
A Servant may respond to a *Read Ready* or *Write Ready* protocol violation by either asserting BERR*, asserting its Err* bit, or not detecting the violation.

OBSERVATION C.3.16:
A Fast Handshake capable Servant prevents incorrect data transfers due to *DIR* and *DOR* violations by asserting BERR*.

RULE C.3.29:
IF a command to a Message Based Device causes the detection of one of the defined word serial protocol errors,
THEN the device **SHALL NOT** execute that command.

RULE C.3.30:
IF a command to a Message Based Device causes the detection of one of the defined word serial protocol errors which does not result in a bus error,
THEN the the Servant **SHALL** clear both the *Err** and *Read Ready* bits. The *Write Ready* bit **SHALL NOT** be set to one (1) until both the *Err** and *Read Ready* bits are cleared to zero (0).

RULE C.3.31:
Each Message Based Device **SHALL** maintain its *Current Error State* (which is reported via the *Read Protocol Error* command) as follows:

1. The current error state is set to *No error* by a HARD RESET, SOFT RESET, or receipt of the *Clear* command.
2. The current error state is set to *No error* when the device responds to the *End Normal Operation* or the *Abort Normal Operation* command.
3. The current error state is set to *No error* after the device responds to the *Read Protocol Error* command.
4. When a word serial protocol error is detected while the device is in the *No Error* state, the current error state is set to reference that error. Subsequent errors **SHALL NOT** cause updates to the error state.

RULE C.3.32:
When a Message Based Servant receives the *Read Protocol Error* command, it **SHALL** set its *Err** bit to one (1) before it sets its *Write Ready* bit to one (1).

OBSERVATION C.3.17:
A Commander that implements rigorous error checking will test for errors after writing each command as follows:

1. Wait for the *Write Ready* bit to be asserted.
2. Test the *Err** bit.
3. If no error is indicated, then read the response data (when *Read Ready* is asserted) or send the next command. Otherwise, send *Read Protocol Error* and read its response.

RECOMMENDATION C.3.4:
Commanders and Resource Managers should implement rigorous error checking when communicating with their Servants, especially during system initialization.

RULE C.3.33:
When a Message Based Servant receives a command requiring it to output a response to its data low register, it **SHALL NOT** make the re-assertion of its *Write Ready* bit contingent on its Commander reading the response data.

C.3.3.5 DEVICE FAILURES

When a device experiences a catastrophic failure, it enters the FAILED state, asserting SYSFAIL* and clearing (to 0) its *Passed* bit.

RULE C.3.34:
IF a device is in the FAILED state,
THEN that device's Commander **SHALL** set (to 1) the *Sysfail Inhibit* bit of the Control register on the failed device.

PERMISSION C.3.5:
When a device fails, the device's Commander **MAY** also set (to 1) the *Reset* bit of the Control register on the failed device.

OBSERVATION C.3.18:

A Commander may detect that a Servant has failed by monitoring the SYSFAIL* signal or polling the *Passed* bit in the Status register of a device.

RECOMMENDATION C.3.5:

When a top level device fails, the Resource Manager should set (to 1) the *Sysfail Inhibit* bit of the Control register on the failed device.

C.4 SYSTEM RESOURCES

The VXIbus architecture provides for a set of common system resources. These include Slot 0 services and system configuration management. This section describes how these services are provided. The configuration management services are provided by a central Resource Manager, and Slot 0 services are either provided by the Resource Manager or by Slot 0 devices.

C.4.1 Resource Manager

The VXIbus Resource Manager is a device at Logical Address 0 that performs the following functions at system power-on.

1. Identify all VXIbus devices in the system.
2. Manage the system self test and diagnostic sequence.
3. Configure the system's A24 and A32 address maps.
4. Configure the system's Commander/Servant hierarchies.
5. Allocate the VMEbus IRQ lines.
6. Initiate normal system operation.

The logical addresses of VXIbus devices can be statically set or dynamically assigned. In this section, the Resource Manager's functions in the case of statically set logical addresses are described. The functions for dynamic assignment of logical addresses are described in Section F, "Dynamic Configuration".

RULE C.4.1:
A Resource Manager **SHALL** be a Message Based Device with Commander capability.

PERMISSION C.4.1:
The functions of a Resource Manager **MAY** be combined with those of another Message Based Commander.

OBSERVATION C.4.1:
If a Resource Manager is combined with other Message Based functionality, the Resource Manager configures its Message Based functions as it would any other Message Based Device. In this case it uses device specific internal communications, instead of word serial commands.

RULE C.4.2:
IF a Resource Manager device is not configured as a top level Commander,
THEN it **SHALL** behave as a normal Message Based Device toward its Commander and Servants.

RULE C.4.3:
A Resource Manager **SHALL** provide a device specific mechanism for reporting system configuration errors.

PERMISSION C.4.2:
A Resource Manager **MAY** provide Slot 0 services in addition to its other duties.

RULE C.4.4:
A Resource Manager which lacks device specific knowledge of a device **SHALL** write a one (1) to all the device dependent bits in the Control register whenever it writes to the Control register of that device.

OBSERVATION C.4.2:
A Resource Manager will normally write to the Control register of a device to affect the *A24/A32 Enable, Sysfail Inhibit*, or *Reset* bits.

C.4.1.1 DEVICE IDENTIFICATION

The Resource Manager identifies the available VXIbus devices according to the following procedure.

RULE C.4.5:
All Resource Managers **SHALL** implement the following device identification procedure.

1. After the de-assertion of SYSRESET*, wait until either SYSFAIL* is de-asserted, or five (5) seconds have elapsed (whichever occurs first) before accessing any other VXIbus device's A16 configuration registers.
2. Attempt a read of the Status register at each of the 256 defined configuration register locations. If an attempt is successful, the corresponding device is present. If a bus error results, the device is absent.

OBSERVATION C.4.3:
Local VMEbus operations that do not access any of the registers of VXIbus devices are permitted before the de-assertion of SYSFAIL* or the five (5) second timeout.

C.4.1.2 SYSTEM SELF TEST MANAGEMENT

RULE C.4.6:
All Resource Managers **SHALL** implement the following self test management procedure.

1. Wait until all self tests have completed.
2. Force all failed devices into the SOFT RESET state with SYSFAIL* assertion inhibited (see Section C.2.1.2, "Device Initialization and Diagnostics").

OBSERVATION C.4.4:
A device has completed its self test when its *Passed* bit is one (1), or when at least five seconds after power-on (Deassertion of SYSRESET*) have elapsed.

OBSERVATION C.4.5:
The Resource Manager forces a device into its SOFT RESET state by writing a one to the device's *Reset* bit.

OBSERVATION C.4.6:
The Resource Manager causes a device to inhibit SYSFAIL* assertion by writing a one to the device's *Sysfail Inhibit* bit.

PERMISSION C.4.3:
A Resource Manager **MAY** perform diagnostic tests of a failed device by some device dependent method.

PERMISSION C.4.4:
A Resource Manager **MAY** implement the capability to detect the subsequent failure of a device that has passed its initial self test.

RULE C.4.7:
When the Resource Manager detects that a device has failed sometime after the initial five seconds but before the Resource Manager has sent the *Begin Normal Operation* command to that device's top-level Commander, the Resource Manager **SHALL** force the failed device into the SOFT RESET state with SYSFAIL* assertion inhibited.

C.4.1.3 ADDRESS MAP CONFIGURATION

RULE C.4.8:
All Resource Managers **SHALL** implement the following A24 and A32 address map configuration procedure:

1. Determine which devices implement A24 or A32 registers by reading the *Address Space* field of each device's ID register.
2. Determine the number of A24 or A32 registers implemented by each device by reading the *Required Memory* field of the device's Device Type register.
3. Calculate A24 and A32 offsets such that no two device's address spaces overlap.
4. Assign the A24 and A32 base addresses by writing the offsets to each device's Offset register.
5. Enable each device's A24 or A32 registers by writing a one (1) to the *A24/A32 Enable* bit of each device's Control register.

RECOMMENDATION C.4.1:
All A24 base address offsets should be assigned so that the registers are placed within the address range 200000_{16} through $DFFFFF_{16}$. All A32 base address offsets should be assigned so that the registers are placed within the address range 20000000_{16} through $DFFFFFFF_{16}$.

C.4.1.4 COMMANDER/SERVANT HIERARCHIES

The Resource Manager establishes a system wide control hierarchy. This structure has the form of one or more inverted trees. The hierarchical relationships are described by the terms Commander and Servant. A Commander is any device in the hierarchy with one or more associated lower level devices. A Servant is any device in the subtree of a Commander. A device may be both a Commander and a Servant in a multiple level hierarchy. A Commander has exclusive control of his immediate Servants' communication and control registers. A top level Commander has no Commander but may have Servants. All Commanders are Message Based Devices.

A Resource Manager establishes the system hierarchy in the following manner.

1. Find all Commanders by checking the *CMDR* bit in the protocol register of each Message Based Device.
2. Read the Servant area size from each Commander, using the *Read Servant Area* query.
3. Determine the Commander/Servant hierarchies.
4. Assign Servants to Commanders, using the *grant device* command.

RULE C.4.9:
A Resource Manager **SHALL** implement some means of building a Commander/Servant hierarchy.

RECOMMENDATION C.4.2:
A Resource Manager should implement the default Commander/Servant mapping algorithm described in the following section.

OBSERVATION C.4.7:
Each Commander maintains a list of its Servants, received via the *grant device* command. It maintains this list until the next activation of HARD RESET or SOFT RESET, or the receipt of the *Abort Normal Operation* command.

RULE C.4.10:
A Resource Manager **SHALL NOT** grant to any Commander a device which the Resource Manager knows to be in any state other than PASSED. (see Section C.2.1.2.2, "Self Test Operation").

OBSERVATION C.4.8:
It can not be assured that the Resource Manager will detect any device failures after the initial self test.

C.4.1.4.1 Commander/Servant Mapping

The default Commander/Servant mapping algorithm allows for a virtually unlimited number of hierarchical levels. It is based on each VXIbus device's Logical Addresses and each Commander's Servant area size. The Servant area size is an 8 bit (0-255) value selected by the user or at the factory. It is stored in a non-volatile fashion in each Commander device.

A Commander's Servant area starts at the next consecutive Logical Address after the Commander's logical address, and includes the number of contiguous Logical Addresses given by the Commander's Servant area size.

If a device is in the Servant area of a given Commander, and it is not in the Servant area of any other device within that Commander's Servant area, then it is a Servant of that Commander, and may be assigned to that Commander by the Resource Manager.

OBSERVATION C.4.9:
All of a device's default Servants are in its own Servant area.

OBSERVATION C.4.10:
A device's Servant area may include the Servant areas of its Servants.

OBSERVATION C.4.11:
None of a device's Servants are located in any of the Servant areas of its Servants.

C.4.1.5 ALLOCATING IRQ LINES

The Resource Manager is responsible for allocating the VMEbus IRQ lines among the various Interrupt Handlers and Interrupters in the system. Each IRQ line may be assigned to only one handler, but to several Interrupters. Provision must be made for devices with jumper selectable IRQ line assignments as well as for programmable Interrupters and Interrupt Handlers. This requires that some interrupt configuration information be supplied by the system user.

RULE C.4.11:
A Resource Manager **SHALL** provide a means for a user to supply interrupt configuration information relating Interrupt Handlers and Interrupters to any or all interrupt lines.

OBSERVATION C.4.12:
The form of the interrupt configuration information is device dependent. It is anticipated that the information will typically be loaded from a system console, downloaded from a host computer, or retrieved from some form of non-volatile storage.

RULE C.4.12:
A Resource Manager **SHALL** allocate IRQ lines to Interrupt Handlers according to the following procedure.

1. Determine which Message Based Devices support Programmable Interrupt Handler (PH) capability (using the *Read Protocol* command).
2. Allocate the IRQ lines according to the following rules:

a. First use the supplied interrupt allocation information.

b. Then allocate one of the remaining IRQ lines to each PH capable Commander, in any desired order.

c. Finally allocate any remaining IRQ lines to any Servant-only PH capable Message Based Devices, and any PH Commanders indicating multiple handler capability.

3. For each Programmable Handler assigned an IRQ line, program its IRQ allocation, using the *Read Handlers, Read Handler Line,* and *Assign Handler Line* commands.

RULE C.4.13:
A Resource Manager **SHALL** allocate IRQ lines to Interrupters according to the following procedure.

1. Determine which Message Based Servants have Programmable Interrupter (PI) capability, using the *Read Protocol* command.

2. Allocate the IRQ lines according to the following rules:

   a. First use the supplied interrupt allocation information.

   b. Then allocate an IRQ line to each PI capable Servant, based on its Commander's IRQ line allocation.

3. For each Programmable Interrupter assigned an IRQ line, program its IRQ allocation, using the *Read Interrupters*, *Read Interrupter Line* and *Assign Interrupter Line* commands.

C.4.1.6 INITIATING NORMAL OPERATION

After allocating IRQ lines, a Resource Manager may provide some system dependent start-up services. It then sends the *Begin Normal Operation* command, with the *Top Level* bit set (to one (1)), to all top level Commanders in order of increasing logical address. At this point the Resource Manager's power-on sequence is complete, and it may enter its run time operating mode.

Each top level Commander is then required to initialize its Servants. Each Commander must send the *Begin Normal Operation* command, with the *Top Level* bit cleared (to zero (0)), to all of its Message Based Servants. Therefore, the *Begin Normal Operation* command propagates down the hierarchy from Commanders to Servants, until all Message Based Devices have received it.

RULE C.4.14:
A Resource Manager **SHALL NOT** change the configuration or state of any other device's Servant from its power-on default conditions, except as required to perform the system self test management, address map configuration, commander/servant mapping, IRQ line allocation, and dynamic Logical Address assignment.

OBSERVATION C.4.13:
When a Commander receives the *Begin Normal Operation* command, its Message Based Servants are in the CONFIGURE sub-state with any Response and Event generation capabilities set according to the default configuration defined in Rule C.2.75.

C.4.2 Runtime Resource Management

In Section C.4.1, "Resource Manager", and in Section F, "Dynamic Configuration", resource management is defined only for default configuration. The behavior of a Resource Manager which provides ongoing services for a device in NORMAL OPERATION sub-state has not been defined. Ongoing resource management services can be used to support shared resource utilization or other system operations. In the following, the term "runtime resource management" is used for such ongoing resource management.

PERMISSION C.4.5:

A Resource Manager which handles default system configuration as described in Section C.4.1, "Resource Manager" (and in section F, "Dynamic Configuration") **MAY** handle runtime resource management.

OBSERVATION C.4.14:

It is a violation of the Word Serial Protocol for a device involved in runtime resource management to send any word serial command (including *End Normal Operation* and *Abort Normal Operation*) to any device while that device's Commander is in the NORMAL OPERATION sub-state.

C.4.3 VXIbus Subsystem Slot 0

A VXIbus Slot 0 device provide common resources to its VXIbus subsystem slots 1-12. On P2, the resources provided are CLK10 and MODID. On P3, the the Slot 0 device provides CLK100. It may also provide SYNC100, STARX, and STARY signal distribution on P3.

OBSERVATION C.4.15:
VXIbus Slot 0 devices exist only on B, C and D form factors.

RULE C.4.15:
All Slot 0 devices **SHALL** supply CLK10.

RULE C.4.16:
All Slot 0 devices **SHALL** support MODID.

RULE C.4.17:
All P3 Slot 0 devices **SHALL** provide CLK100.

OBSERVATION C.4.16:
SYNC100, STARX, and STARY are not required to be supported by Slot 0 devices.

RECOMMENDATION C.4.3:
If only one STAR signal set is provided, it should be STARX.

RECOMMENDATION C.4.4:
IF a Slot 0 device supports either STAR signal set
THEN it should support a contiguous set of slots including slot 1.

OBSERVATION C.4.17:
A Slot 0 device is only guaranteed to provide CLK10 and MODID support. It may also provide other services such as SYNC100 arbitration, trigger line buffering, and STARX and STARY routing.

A Slot 0 device may be identified by its *Model Code*, which has a value in the range 0 → 255 (0_{16} → FF_{16}).

RULE C.4.18:
All VXIbus Slot 0 devices **SHALL** have model codes in the range 0_{16} → FF_{16}.

RULE C.4.19:
A VXIbus device which is not enabled for Slot 0 functionality **SHALL NOT** have a model code in the range 0_{16} → FF_{16}. A Slot 0 device with its Slot 0 functions disabled (via switches, jumpers, cut traces, etc) **SHALL** also have its *Model Code* modified to some value outside the range 0_{16} → FF_{16}.

RECOMMENDATION C.4.5:
A Slot 0 module's Slot 0 and VMEbus system controller capability should be defeatable in order to allow that module to be placed in any slot position.

RULE C.4.20:
All MODID drivers **SHALL** be disabled by both hard and soft resets (See Section C.2.1.2.2, "Self Test Operation").

C.4.3.1 REGISTER BASED SLOT 0 DEVICES

A Register Based Slot 0 Device is a standard VXIbus Register Based Device that supports Slot 0 functionality. Its *Model Code* (in its Device Type register - See Section C.2.1.1.2, "Configuration Registers") is in the range 0 → 255 (0_{16} → FF_{16}), and it has a MODID register at its A16 base address + 8_{16}. It may implement additional Slot 0 related registers.

RULE C.4.21:
A Register Based VXIbus Slot 0 Device **SHALL** implement a MODID register as defined in the following section (C.4.3.1.1) at its A16 base address + 8_{16}.

PERMISSION C.4.6:
The remaining device dependent registers **MAY** be used to implement other Slot 0 related services.

C.4.3.1.1 MODID Register

The MODID register indicates the status of and provides control of the associated MODID lines. It conforms to the following format.

Bit #	15 ← 14	13	12 ← 0
Contents	RESERVED	Output Enable	MODID00-12

The fields of the MODID register are defined below.

- RESERVED: This field is reserved for future VXIbus definition. A write to this field has no effect. A read of this field returns all ones (3_{16}).
- Output Enable: Writing a one (1) to this bit enables the Slot 0 MODID drivers. Writing a zero (0) to this bit disables the MODID drivers. A one (1) read from this bit indicates that the MODID drivers are enabled. A zero (0) indicates that the drivers are disabled.

  OBSERVATION C.4.18:
  All MODID drivers are disabled by device resets (The *Output Enable* bit is cleared (to 0)).
- MODID 00-12: Writing a one (1) to any of these bits drives the corresponding MODID line high, if the *Output Enable* bit is set. Writing a zero (0) to any of these bits drives the corresponding line low, if the *Output Enable* bit is set. When read, each of these bits indicates the actual level of the corresponding MODID line (1 = HI, 0 = LOW).

  OBSERVATION C.4.19:
  After system power up, the bit values of MODID00-12 in the MODID write register may be undefined.

C.4.3.2 MESSAGE BASED SLOT 0 DEVICES

A Message Based Slot 0 Device is a standard VXIbus Message Based Device that supports Slot 0 functionality. Its *Model Code* (in its Device Type register - See Section C.2.1.1.2, "Configuration Registers") is in the range 0 → 255, and it supports the MODID related commands. It may implement additional Slot 0 related commands.

RULE C.4.22:
A Message Based VXIbus Slot 0 Device **SHALL** implement the *Read MODID*, *Set Lower MODID*, and *Set Upper MODID* commands as defined in Section E.1, "Word Serial Commands".

PERMISSION C.4.7:
A Message Based Slot 0 Device **MAY** implement commands to provide other Slot 0 related services.

The MODID related commands are used to indicate the status of the MODID lines and to control the levels of the MODID lines.

OBSERVATION C.4.20:
All MODID drivers are disabled by device resets.

C.4.3.3 OTHER SLOT 0 DEVICES

Since VXIbus only requires support of MODID and CLK10 lines from a Slot 0 device, the Resource Manager may provide the Slot 0 services in addition to its other functions. This arrangement hides the Slot 0 functionality inside the Resource Manager and therefore allows another device to provide Slot 0 functionality without using a Logical Address for a minimal Slot 0 implementation.

D. VXIbus DEVICE IMPLEMENTATIONS

This section describes the design rules for certain devices which facilitate manufacturer to manufacturer interchangeability. The protocols are defined at a higher level than defined by the system architecture. These protocols, along with the common command formats are specified in the following sections.

One common device is a VXIbus to IEEE 488 bus [2] interface. This device allows VXIbus devices to be controlled via the 488 bus as if they are 488 devices. The following sections define such an interface, its VXIbus protocols, and the requirements for an instrument to interface to it. The protocols are intended to be extensible to other interfaces and Commanders.

D.1 VXIbus INSTRUMENTS

This section specifies the interfacing requirements for **VXIbus Instruments**. These are devices which may be controlled with the **VXIbus Instrument Protocol**, implemented by a **VXIbus Interface Device**.

D.1.1 VXIbus Instrument Protocols

RULE D.1.1:
A VXIbus instrument **SHALL** be a Message Based Device and provide the VXIbus word serial communication protocol.

VXIbus instruments communicate with the core set of commands and events described in the following sections.

OBSERVATION D.1.1:
The following commands are considered to be Instrument Protocol commands:

- *Byte Available*
- *Byte Request*
- *Clear*
- *Trigger*
- *Set Lock*
- *Clear Lock*
- *Read STB*

D.1.1.1 DATA TRANSFER from COMMANDERS to INSTRUMENTS

The *Byte Available* command is used to send a data byte (DAB) to a VXIbus instrument. The command includes an END bit which may be used as a message terminator.

RULE D.1.2:
All VXIbus instruments **SHALL** support the *Byte Available* command in conformance to the Byte Transfer Protocol.

OBSERVATION D.1.2:
The Byte Transfer Protocol provides for an instrument to hold off incoming *Byte Available* commands while still allowing other commands such as *Clear* and *Read STB*.

OBSERVATION D.1.3:
The state of *DIR* when the *Write Ready* bit becomes asserted allows a Commander to determine whether or not to send a *Byte Available* or *Trigger* command to the Servant. If *DIR* is set, then the Commander may send a *Byte Available* or *Trigger* command.

D.1.1.2 DATA TRANSFER from INSTRUMENTS to COMMANDERS

Data transfer from an instrument to its Commander is initiated by the Commander by sending a *Byte Request* command to the VXIbus instrument. The instrument responds by making the data byte available in its data registers. The returned data includes an END bit which may be used as a message terminator.

RULE D.1.3:
All VXIbus instruments **SHALL** support the *Byte Request* command in conformance to the Byte Transfer Protocol.

OBSERVATION D.1.4:
The Byte Transfer Protocol provides for an instrument to indicate the presence of data in its output buffer (thus readiness for a *Byte Request* command).

OBSERVATION D.1.5:
The state of *DOR* when the *Write Ready* bit becomes asserted allows a Commander to determine whether or not to send a *Byte Request* command to the Servant. If *DOR* is set, then the Commander can send the *Byte Request* command.

D.1.1.3 CLEARING a VXIbus INSTRUMENT

A VXIbus instrument which receives a *Clear* command will respond according to the following rules (in addition to Rules C.2.75 and C.2.76).

RULE D.1.4:
Upon receipt of the *Clear* command the VXIbus instrument **SHALL** perform the following actions:

- clear its input buffer (set *DIR*)
- clear its output queue (clear *DOR*, if appropriate)
- become ready to process new commands

Further clarification of the proper operation of the *Clear* command may be found in the IEEE 488.2 Standard [3], Section 5.8, "Device Clear Requirements".

OBSERVATION D.1.6:
The VXIbus *Clear* command may affect current instrument operation for VXIbus instruments, while for VXIbus IEEE 488.2 instruments it will not.

RULE D.1.5:
All VXIbus instruments **SHALL** support the *Clear* command.

D.1.1.4 TRIGGERING an INSTRUMENT

RULE D.1.6:
All VXIbus instruments which support a trigger function **SHALL** be capable of being triggered via the *Trigger* command.

PERMISSION D.1.1:
VXIbus Instruments may support other methods of triggering (i.e. P2 or P3 hardware trigger busses).

RULE D.1.7:
When a VXIbus instrument is not ready to process a *Trigger* command it **SHALL** clear to zero (0) the *DIR* bit in its Response register.

OBSERVATION D.1.7:
When a VXIbus instrument is ready to process a *Trigger* command it will normally set to one (1) the *DIR* bit in its Response register.

D.1.1.5 LOCAL LOCKOUT

A Commander can issue the *Set Lock* command to its Servant if it wishes the Servant to ignore commands from local sources. The *Clear Lock* command is used by a Commander to cause a device to exit the Lock state.

RULE D.1.8:
IF an instrument supports the *Set Lock* command,
THEN upon receipt of the *Set lock* command it **SHALL** clear (to zero (0)) the *Locked** bit (bit 7) in its Response register.

RULE D.1.9:
IF an instrument supports the *Clear Lock* command,
THEN upon receipt of the *Clear lock* command it **SHALL** set (to one (1)) the *Locked** bit (bit 7) in its Response register.

RULE D.1.10:
IF an instrument supports the *Set Lock* command,
THEN local control of the device **SHALL** be disabled upon the receipt of a *Set Lock* command.

RULE D.1.11:
IF an instrument has had local control disabled by the receipt of a *Set Lock* command,
THEN local control of that device **SHALL NOT** be re-enabled until the receipt of a *Clear Lock* command.

PERMISSION D.1.2:
Local operations which do not affect device operations will be allowed while the device is in the locked state as per IEEE 488.2.

PERMISSION D.1.3:
VXIbus instruments **MAY** implement the *Set Lock* and *Clear Lock* commands.

RECOMMENDATION D.1.1:
If an instrument implements either the *Set Lock* or the *Clear Lock* command then it should implement both commands.

D.1.1.6 SRQ OPERATION

When a device needs to request service on an interface, it sends a *request true* event to its Commander. This event can be sent via the event form of the interrupt response word or as a signal. If the need for service is removed, a device which has previously sent a *request true* event must send the *request false* event. This event is sent in an analogous manner to the *request true* event.

PERMISSION D.1.4:
VXIbus instruments **MAY** implement the *request true* and *request false* events.

RULE D.1.12:
IF a VXIbus Instrument has VMEbus Master capability
THEN the *request true* and *request false* events **SHALL** be sent using signals (unless modified by the *Asynchronous Mode Control* command).

RULE D.1.13:
IF a VXIbus Instrument does not have VMEbus Master capability
THEN the *request true* and *request false* events **SHALL** be sent using interrupts (unless modified by the *Asynchronous Mode Control* command).

D.1.1.7 SPOLL OPERATION

PERMISSION D.1.5:
VXIbus instruments **MAY** implement the *Read STB* command.

RULE D.1.14:
A VXIbus instrument which implements the *Read STB* command **SHALL** send as bit 6 of it's response a bit that is equivalent to the MSS bit of the *STB? response in IEEE Standard 488.2-1987.

OBSERVATION D.1.8:
This bit is to indicate whether or not the device is currently requesting service. In an interface other than IEEE 488.1 the RQS function may not be available. In this case, the device indicates it's current needs to the requester via this bit.

D.1.1.8 ERROR REPORTING

Error reporting is divided into two parts: Word Serial Protocol errors, and errors resulting from messages sent via the *Byte Available* command. Word Serial Protocol error reporting is described in Section C.3.3.4, "Error Handling". Errors which are the result of higher-level messages received via the *Byte Available* command are reported via the *request true* event.

RULE D.1.15:
IF a device reports errors via signals or interrupts as a result of messages received via the *Byte Available* command,
THEN that device **SHALL** use the *request true* event to notify its Commander of the error.

RECOMMENDATION D.1.2:
A VXIbus Instrument should follow the IEEE 488.2 error reporting conventions (Reference: IEEE 488.2, Sections 11.3 and 11.5) including the enabling and disabling of the *request true* event generation.

D.1.1.9 INITIALIZATION

VXIbus instruments require no initialization beyond that required for all Message Based Instruments. Once a VXIbus system is configured, the instruments are ready to process Instrument Protocol commands.

RULE D.1.16:
After the receipt of the *Begin Normal Operation* command, a VXIbus Instrument **SHALL** become ready to accept Instrument protocol commands.

OBSERVATION D.1.9:
Normally when a device is ready to accept Instrument protocol commands it will have the following indications: *DIR* = 1, *DOR* = 0, *Locked* = 1, *Read Ready* = 0, *Write Ready* = 1, and *Err** = 1.

OBSERVATION D.1.10:
A device which implements a self initiated trigger, may have data ready for output when it is ready to accept Instrument protocol commands. It will indicate this with *DOR* = 1.

D.1.2 VXIbus IEEE-488.2 Instrument Protocols

This section describes additions to the VXIbus Instrument Protocols which apply to VXIbus IEEE-488.2 Instruments.

RULE D.1.17:
All VXIbus IEEE-488.2 Instruments **SHALL** comply with all the rules for VXIbus Instruments.

D.1.2.1 CLEARING a VXIbus IEEE-488.2 INSTRUMENT

RULE D.1.18:
The *Clear* command **SHALL NOT** affect current instrument operation for a VXIbus IEEE-488.2 Instrument.

RULE D.1.19:
VXIbus IEEE-488.2 Instruments **SHALL** implement the *Clear* command according to the specifications in the IEEE 488.2 Standard, Section 5.8, "Device Clear Requirements".

D.1.2.2 TRIGGERING an IEEE-488.2 INSTRUMENT

OBSERVATION D.1.11:
VXIbus IEEE 488.2 instruments **MAY** support either DT0 or DT1 IEEE 488.1 subsets.

D.1.2.3 LOCAL LOCKOUT

OBSERVATION D.1.12:
VXIbus IEEE 488.2 instruments **MAY** support either RL0 or RL1 IEEE 488.1 subsets.

D.1.2.4 SRQ OPERATION

RULE D.1.20:
All VXIbus IEEE 488.2 instruments **SHALL** implement the *request true* and *request false* events.

D.1.2.5 SPOLL OPERATION

RULE D.1.21:
All VXIbus IEEE 488.2 instruments **SHALL** implement all the required 488.2 status reporting registers and protocols and the *Read STB* command.

OBSERVATION D.1.13:
The response to a *Read STB* command will have a bit 6 which is equivalent to the MSS bit of the STB? response in IEEE Standard 488.2-1987.

D.2 488-VXIbus INTERFACE

The 488-VXIbus interface device converts IEEE 488 protocols to VXIbus instrument control protocols, allowing IEEE 488 external controllers to control VXIbus instruments in a manner analogous to a typical rack and stack instrument system. The primary function of the 488-VXIbus Interface device is to route the IEEE 488 messages to the appropriate VXIbus instrument and to return the results from the instrument to the external IEEE 488 controller. A VXIbus system may or may not provide one or more 488-VXIbus Interface devices.

RULE D.2.1:
A 488-VXIbus Interface **SHALL** be a Message Based Commander.

RULE D.2.2:
A 488-VXIbus Interface device **SHALL** support the receipt of signals.

RULE D.2.3:
A 488-VXIbus Interface device **SHALL** be an Interrupt Handler.

PERMISSION D.2.1:
When a 488-VXIbus Interface device detects that a Servant has not recognized a command (error response from the Servant), the interface action is left to the device designer.

D.2.1 IEEE 488 Address Mapping

The mapping of an IEEE 488 address to VXIbus Logical Address is a function which is handled by the 488-VXIbus Interface device. The particulars of this mapping are interface device specific. Several potential mappings are described below:

- The 488-VXIbus Interface device supports primary addressing of 31 potential addresses. The 31 IEEE 488 addresses will be associated one for one with 31 Logical Addresses of the VXIbus instruments.
- The 488-VXIbus Interface device supports secondary addressing of 31 potential secondary addresses. The 488-VXIbus Interface device will provide a means of selecting the primary address. The 31 IEEE 488 secondary addresses will be associated one for one with 31 Logical Addresses of the VXIbus instruments.
- The 488-VXIbus Interface device supports a single primary IEEE 488 address through which all VXIbus Instruments are addressed. The particular language constructs which implement routing are up to the interface device designer.

OBSERVATION D.2.1:
488-VXIbus Interface devices can use these or any other address mapping scheme.

OBSERVATION D.2.2:
488-VXIbus Interface devices which implement different address mapping schemes might require address changes for portability. The application which controls VXIbus instruments must consider the address mapping.

D.2.2 488-VXIbus Interface Device to IEEE 488 Bus Functions.

A 488-VXIbus interface device will provide a set of capabilities which enables a VXIbus instrument to comply with IEEE 488.2. The actual choice of whether or not a VXIbus instrument will implement IEEE 488.2 is left to the instrument designer.

RULE D.2.4:
488-VXIbus Interface device **SHALL** implement the following IEEE 488.1 subsets, in an IEEE 488.2 compatible fashion, for each VXIbus Instrument it controls:

- SH1 Source Handshake.
- AH1 Acceptor Handshake.
- T6 Addressing Talker or TE6 Extended Addressing Talker - (No Talk Only Mode).
- L4 Listener or LE4 Extended Listener - (No Listen Only Mode).
- SR1 Service Request.
- DC1 Device Clear.
- DT1 Device Trigger.

OBSERVATION D.2.3:
The preceding rule guarantees that a VXIbus Instrument which is compatible with IEEE 488.2 will be IEEE 488.2 compatible when accessed through the 488-VXIbus interface device.

PERMISSION D.2.2:
RL1 Remote/Local capability **MAY** be implemented by the 488-VXIbus Interface device.

PERMISSION D.2.3:
Controller capability **MAY** be implemented by the 488-VXIbus Interface device.

RULE D.2.5:
IF a 488-VXIbus Interface device supports controller capability,
THEN the controller **SHALL** follow the IEEE 488.2 implementation.

RECOMMENDATION D.2.1:
488-VXIbus Interface devices should implement E2 three-state electrical drivers.

RULE D.2.6:
IF a 488-VXIbus Interface device implements parallel poll capability
THEN it **SHALL** be implemented according to IEEE 488.2 rules (PP1).

RULE D.2.7:
The 488-VXIbus Interface device **SHALL** preserve the order of received messages, including data bytes (DAB), END messages, Group Execute Trigger, Device Clear, and Selective Device Clear.

D.2.3 VXIbus Instrument Protocol

RULE D.2.8:
VXIbus Instruments and VXIbus Interface devices **SHALL** be logically partitioned according to Figure D.1.

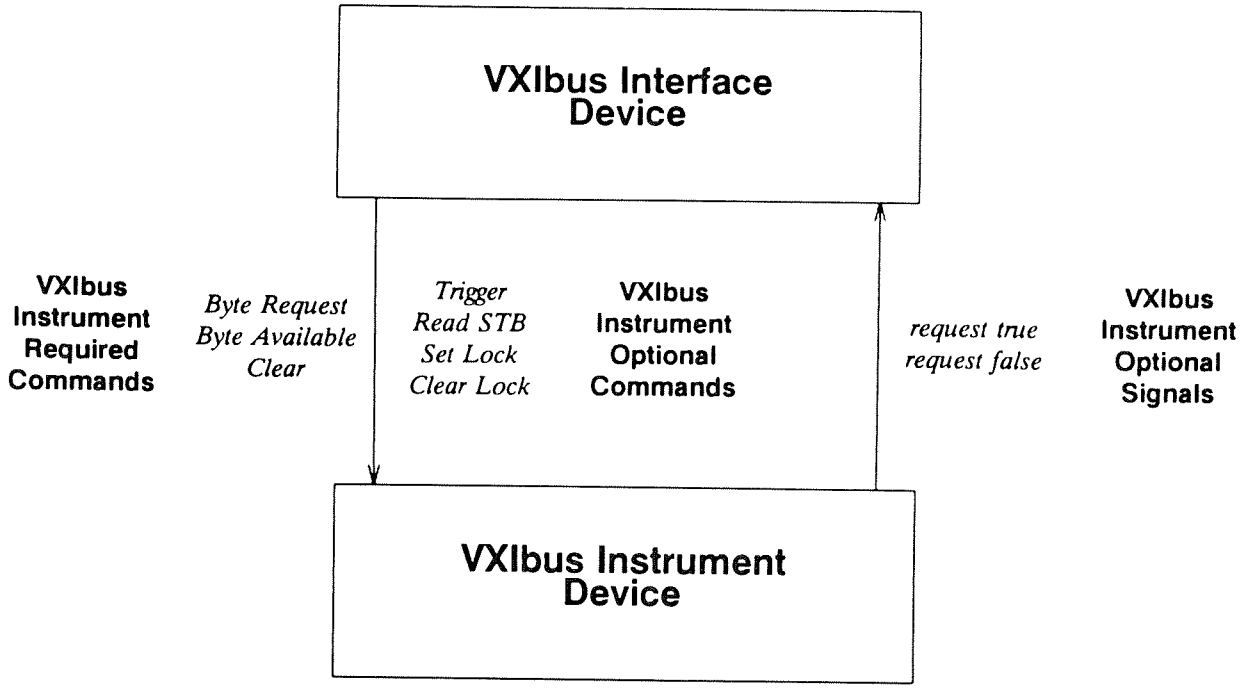


Figure D.1. VXIbus Instrument Protocol

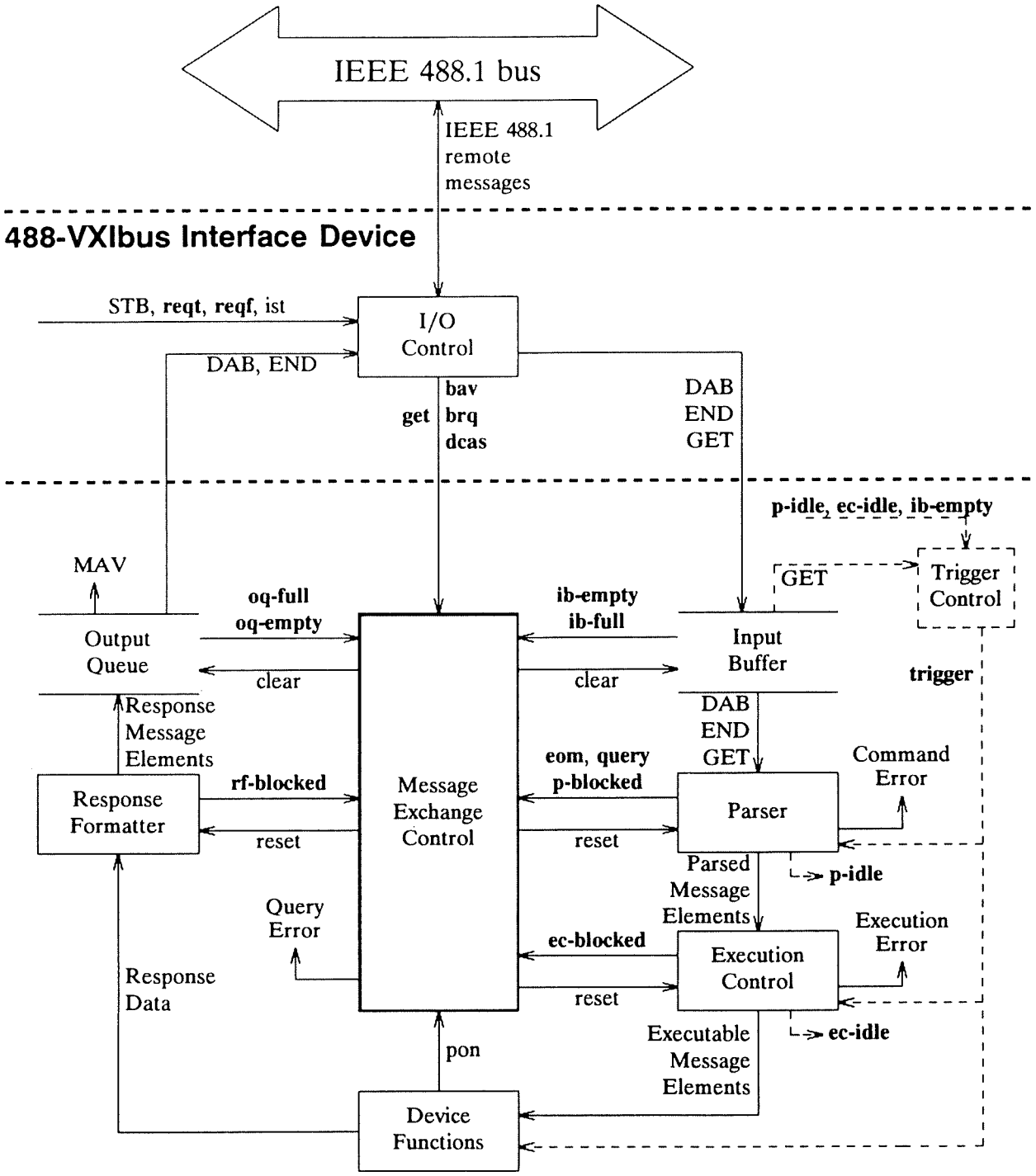
RULE D.2.9:
IEEE 488.2 compatible VXIbus Instruments **SHALL** be logically partitioned according to Figure D.2.

D.2.3.1 DATA TRANSFER from INTERFACE DEVICE to VXIbus INSTRUMENT

RULE D.2.10:
IF EOI is asserted on receipt of a data byte from the external interface,
THEN the interface device **SHALL** send the *Byte Available* command with END asserted to all Servants which are addressed to listen (LACS).
ELSE the interface device **SHALL** send the *Byte Available* command with END unasserted to all Servants which are addressed to listen (LACS).

OBSERVATION D.2.4:
A VXIbus Interface device cannot send END for message terminators that do not have EOI asserted.

OBSERVATION D.2.5:
A VXIbus Interface device can not send a *Byte Available* command to an instrument which has its *DIR* bit cleared to zero (0).



IEEE 488.2 Compatible VXIbus Instrument

Figure D.2. Message Exchange Control Interface Functional Blocks

D.2.3.2 DATA TRANSFER from VXIbus INSTRUMENT to VXIbus INTERFACE DEVICE

RULE D.2.11:
When an interface device receives a byte request from the external interface it **SHALL** perform the following actions on the VXIbus:

- The interface device sends the *Byte Request* command to the Servant device which is currently addressed to talk (TACS).
- The interface device then reads the result of the *Byte Request* command from the appropriate Servant device using the word serial protocol.
- The interface places the DAB portion of the response on external interface. Simultaneously, the interface **SHALL** assert the EOI line if the END bit in the response is asserted; else the interface **SHALL NOT** assert EOI.

OBSERVATION D.2.6:
A VXIbus Interface device can not send a *Byte Request* command to an instrument which has its *DOR* bit cleared to zero (0).

D.2.3.3 DEVICE CLEAR OPERATION

This section describes the operation of a 488-VXIbus Interface device when it receives an IEEE 488 device clear message.

RULE D.2.12:
The 488-VXIbus Interface device **SHALL** send the *Clear* command to all VXIbus instruments which it controls when the 488-VXIbus interface device receives the 488.1 DCL command.

RULE D.2.13:
The 488-VXIbus Interface device **SHALL** send the *Clear* command to all VXIbus instruments which are addressed to listen and which it controls when the 488-VXIbus interface device receives the 488.1 SDC command.

RULE D.2.14:
The 488-VXIbus Interface device **SHALL** hold off NDAC on the IEEE 488 interface until the *Clear* command has been sent to all affected devices.

RULE D.2.15:
The 488-VXIbus interface **SHALL NOT** perform a soft reset of any VXIbus instruments in response to any 488.1 defined commands or messages, such as DCL, SDC, or IFC.

D.2.3.4 TRIGGER OPERATION

This section describes the operation of a 488-VXIbus Interface device when it receives an IEEE 488 trigger message.

RULE D.2.16:
Upon receipt of the 488.1 GET message, the 488-VXIbus Interface device **SHALL** send the *Trigger* command to all VXIbus instruments which

- are addressed to listen,
- and are controlled by that particular interface device,
- and implement the *Trigger* command.

- and are not configured for triggering via an alternate trigger implementation.

OBSERVATION D.2.7:
The above rule allows an interface which knows that some Servants provide alternate trigger implementations to be triggered via that implementation.

RULE D.2.17:
The 488-VXIbus Interface device **SHALL** hold off NDAC on the IEEE 488 interface until the *Trigger* command has been sent to all affected devices.

OBSERVATION D.2.8:
There will be a time delay between the receipt of the *Trigger* command by the various VXIbus instruments in the VXIbus system.

OBSERVATION D.2.9:
The possible inclusion of P1 only VXIbus devices in the system mandates a *Trigger* command rather than use of a trigger line.

RULE D.2.18:
A VXIbus Interface device **SHALL NOT** send a *Trigger* command to an instrument which has its *DIR* bit cleared to zero (0).

D.2.3.5 IEEE 488 REMOTE LOCAL

This section discusses the implementation of the IEEE-488 RL1 functions within the context of a VXIbus system. Local is defined as operations which are initiated from any non-VXIbus device interface.

D.2.3.5.1 Locking Devices

The 488-VXIbus Interface will issue the *Set Lock* and *Clear Lock* commands according the the remote/local state of the associated device as maintained by the state table according to the rules below.

RULE D.2.19:
IF A 488-VXIbus Interface Device implements RL1,
THEN it **SHALL** issue the *Set Lock* command to a device which implements the *Set Lock* command when the interface detects that device should have entered the RWLS state of the IEEE-488.1.

RULE D.2.20:
IF A 488-VXIbus Interface Device implements RL1,
THEN it **SHALL** issue the *Clear Lock* command to a device which implements the *Clear Lock* command when the interface detects that device should have exited the RWLS state of the IEEE-488.1.

D.2.3.6 SRQ OPERATION

This section describes the 488-VXIbus Interface device requirements for support of the IEEE 488.2 SRQ capability. 488.2 simplifies the implementation of service request by requiring two synchronization messages which control the assertion of the 488.1 rsv message. Two messages are sourced by the device: the reqt is used to assert a 488.1 rsv (requires service message) and the reqf is used to asynchronously remove rsv. In a VXIbus system these messages are sent using the *request true* and *request false* events. The 488.2 synchronization protocols allow the interface device to:

- Assert SRQ if a previously "enabled" condition occurs.

- Keep SRQ asserted until the controller has recognized the service request and serial polled the device, or the instrument has removed the request (by sending a *request false* event).
- Release SRQ when serial polled so that the controller can detect a SRQ from another instrument.
- Assert SRQ again if another condition occurs whether or not the controller has cleared the first condition.

For further clarification of SRQ and serial poll operation of the 488-VXIbus Interface device, see IEEE 488.2, Section 11.3.3, "Service Request Generation".

RULE D.2.21:
A 488-VXIbus Interface device, upon receipt of a *request true* event from a VXIbus instrument, **SHALL** set the 488.1 rsv message true.

RULE D.2.22:
A 488-VXIbus Interface device, upon receipt of a *request false* event from a VXIbus instrument, **SHALL** set the 488.1 rsv message false only if no *request true* has been received from any other VXIbus Instrument.

OBSERVATION D.2.10:
The VXIbus interface device has to keep track of requests for each of its Servants in order to properly control the SRQ line on the IEEE 488.1 interface.

RULE D.2.23:
A 488-VXIbus Interface device **SHALL** implement the IEEE 488.2 SRQ protocols.

OBSERVATION D.2.11:
Request true and *request false* events may be sent via either interrupts or signals. The 488-VXIbus Interface device is required to accept these events regardless of method of transmission.

D.2.3.7 SPOLL Operation

This section discusses the implementation of the IEEE 488 Serial Poll function within the context of a VXIbus system. The IEEE 488 specific functions, such as whether the device is in SPAS, are managed by the IEEE 488 VXIbus Interface device. When the IEEE 488-VXIbus interface device is ready to source the serial poll response to the 488 interface, it sends a *Read STB* command to the device for which the serial poll was requested. It then reads (using the word serial protocol) the status byte of the polled instrument, and makes this byte available to the interface as the serial poll response.

RULE D.2.24:
IF a VXIbus instrument supports the *Read STB* command,
THEN the IEEE 488-VXIbus Interface device **SHALL** respond to a serial poll of that instrument with the status byte obtained by the *Read STB* command.

RULE D.2.25:
IF a VXIbus instrument doesn't support the *Read STB* command
THEN the IEEE 488-VXIbus Interface device **SHALL** respond to a serial poll of that instrument with bit 6 of the serial poll response byte properly indicating the rsv state of the instrument. Bit 6 will be set (to one (1)) if the instrument is requesting service, otherwise it is cleared (to zero (0)).

RULE D.2.26:
A IEEE 488-VXIbus Interface device **SHALL** implement SPOLL in the manner specified by IEEE 488.2.

E. COMMAND and EVENT FORMATS

This section summarizes the required and optional commands and events which support the instrument protocols.

E.1 WORD SERIAL COMMANDS

The following commands are the standard set of VXIbus device commands which are sent with the word serial protocol.

RULE E.1.1:

Message Based Devices **SHALL** implement the appropriate subsets of the following command set as specified in Table E.1 and E.2.

	Primary Classification			CMDR	MASTER	Slot 0	Comments
	All MBD	I	I4				
Abort Normal Operation	R	R	R	R	R	R	
Begin Normal Operation	R	R	R	R	R	R	
Byte Available		R	R				Requires *DIR* bit
Byte Request		R	R				Requires *DOR* bit
Clear	R	R	R	R	R	R	
Clear Lock							Required with RL 1 Requires *Locked** bit
End Normal Operation	R	R	R	R	R	R	
Grant Device				R			
Identify Commander				R	R		
Read MODID						R	
Read Protocol	R	R	R	R	R	R	
Read Protocol Error	R	R	R	R	R	R	
Read STB			R				
Read Servant Area				R			
Release Device				R			
Set Lock							Required with RL 1 Requires *Locked** bit
Set Lower MODID						R	
Set Upper MODID						R	
Trigger							Optional for all devices Requires *DIR* bit

R - Required for this type of device.

TABLE E.1. General Command Requirements

	Programmable Handler	Programmable Interrupter	Response Generation	Event Generation	Comments
Assign Handler Line	R				
Assign Interrupter Line		R			
Asynchronous Mode Control			R	R	
Control Event				R	
Control Response			R		
Read Handler Line	R				
Read Handlers	R				
Read Interrupter Line		R			
Read Interrupters		R			

R - Required for this type of device.

TABLE E.2. Asynchronous Command Requirements

The following is a list of commands:

Abort Normal Operation: The *Abort Normal Operation* command is used to cause a device to cease normal operation. Upon receipt of this command the device returns to its default configuration, aborting all operations. The aborted state is defined as follows: All VMEbus Master activity is halted. Pending interrupts are unasserted. No new bus requests or interrupts may be asserted. The device is in a generally inactive state and is ready to accept commands.

The syntax of the *Abort Normal Operation* command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1

When the abort operation has completed (the device is in the aborted state), response data is placed in the Data Low register in the following format:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Assign Handler Line: The *Assign Handler Line* command is used to assign a VMEbus IRQ line to a particular Interrupt Handler in the Servant device. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	X	Hand_ID			X	Line		

- X: Don't care. The value written to this bit has no effect.
- Hand_ID: This is a unique identifier of the particular Interrupt Handler being assigned. It has a range of 1 to 7. Within a device, multiple handlers are identified in consecutive order, starting at 1.
- Line: This is the VMEbus IRQ line number. A value of zero (0_{16}) indicates that the handler is to be disconnected.

When the assignment operation has completed, response data is placed in the Data Low register in the following format:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				1	1	1	1	1	1	1	1	1	1	1	0

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The handler referenced in the Hand_ID field is unknown to this device.

OBSERVATION E.1.1:

There is no implicit relationship between the Handler ID (Hand_ID) and the functionality of the associated handler.

Assign Interrupter Line: The *Assign Interrupter Line* command is used to assign a VMEbus IRQ line to a particular Interrupter in the Servant device. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	0	X	Int_ID			X	Line		

- X: Don't care. The value written to this bit has no effect.
- Int_ID: This is a unique identifier of the particular Interrupter being assigned. It has a range of 1 to 7. Within a device, multiple Interrupters are identified in consecutive order, starting at 1.
- Line: This is the VMEbus IRQ line number. A value of zero (0_{16}) indicates that the Interrupter is to be disconnected.

When the assignment operation has completed, response data is placed in the Data Low register in the following format:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				1	1	1	1	1	1	1	1	1	1	1	0

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The handler referenced in the Int_ID field is unknown to this device.

OBSERVATION E.1.2:
There is no implicit relationship between the Interrupter ID (Int_ID) and the functionality of the associated Interrupter.

Asynchronous Mode Control: The *Asynchronous Mode Control* command is used by a Commander to direct the path of events and responses. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	0	1	0	0	0	X				Resp. En*	Event En*	Resp. Mode	Event Mode

The bits have the following meanings:

- X: Don't care. The value written to this bit has no effect.
- Resp. En*: A zero (0) enables generation of responses. A one (1) disables generation of responses.
- Event En*: A zero (0) enables generation of events. A one (1) disables generation of events.
- Resp. Mode: A one (1) indicates that responses should be sent as signals. A zero (0) indicates that responses should be sent as interrupts. This bit is meaningful only if the *Resp En** bit is zero (0).
- Event Mode: A one (1) indicates that events should be sent as signals. A zero (0) indicates that events should be sent as interrupts. This bit is meaningful only if the *Event En** bit is zero (0).

The result data is placed in the Data Low register with the following format. This result is a confirmation/denial of the command.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Status				1	1	1	1	1	1	1	1	Resp. En*	Event En*	Resp. Mode	Event Mode

The bits have the following meanings:

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed as expected.
  - 7_{16}: Command failed - A requested option is not supported.
- Resp. En*: A zero (0) indicates that the generation of responses is enabled. A one (1) indicates that the generation of responses is disabled.
- Event En*: A zero (0) indicates that the generation of events is enabled. A one (1) indicates that the generation of events is disabled.
- Resp. Mode: A one (1) indicates that responses are being sent as signals. A zero (0) indicates that responses are being sent as interrupts. This bit is meaningful only if the *Resp En** bit is zero (0).
- Event Mode: A one (1) indicates that events are being sent as signals. A zero (0) indicates that events are being sent as interrupts. This bit is meaningful only if the *Event En** bit is zero (0).

Begin Normal Operation: The *Begin Normal Operation* command notifies the device that it can begin normal operation now. A Commander may not initiate any VMEbus operations after power-up until it receives the *Begin Normal Operation* command. It then sends the *Begin Normal Operation* command to all of its current Message Based Servants and begins normal execution of commands and/or application programs. The *Top_Level* field of the *Begin Normal Operation* command is provided to inform a device whether or not it is a top level Commander. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	0	Top Level	1	1	1	1	1	1	1	1

- Top Level: A one (1) in this field indicates that the device is a top level Commander. A zero (0) indicates that that it is a Servant to another device.

When the begin operation has completed, response data is placed in the Data Low register in the following format:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				State				Logical Address							

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The Begin Normal Operation command has been successfully executed. The value FE_{16} is reported in the Logical Address field.
  - 7_{16}: The Commander device is not able to command a Servant. The Servant's Logical Address is reported in the Logical Address field.
  - 6_{16}: The Commander device is not able to configure a Servant device. The Servants Logical Address is reported in the Logical Address field.
  - 5_{16}: The Commander device timed out when waiting for response from a Servant device. The Servant's Logical Address is reported in the Logical Address field.
  - 4_{16}: The device is not able to successfully initialize itself. The value FE_{16} is reported in the Logical Address field.
  - 3_{16}: The device is not able to be a top level Commander. The value FE_{16} is reported in the Logical Address field.
  - 2_{16}: The device must be a top level Commander. The value FE_{16} is reported in the Logical Address field.
  - 1_{16}: An undefined error was caused. The value FE_{16} is reported in the Logical Address field.
- State: This field indicates the state of this device and its sub-tree. It may have the following values:
  - 3_{16}: This device and all of its Servant tree are in the CONFIGURE sub-state.
  - 7_{16}: This device is in the CONFIGURE sub-state, however at least one device in its Servant tree is in the NORMAL OPERATION sub-state.
  - B_{16}: This device is in the NORMAL OPERATION sub-state, however at least one device in its Servant tree is in the CONFIGURE sub-state.

– F_{16}: This device and all of its Servant tree are in the NORMAL OPERATION sub-state.

- Logical Address: This field contains the Logical Address corresponding to the status field values.

Byte Available: The *Byte Available* command is used by a Commander to send a byte of data to a Servant device. The END field identifies that this is the last byte of the message. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	1	1	0	END	Datum							

Byte Request: The *Byte Request* command is used by a Commander to read a byte of data from a Servant device. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1

The result data is placed in the Data Low register with the following format. The END field is used to indicate the last byte of the message.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	END	Datum							

Clear: The *Clear* command is used by a Commander to cause a Servant device to clear the VXIbus interface and any pending operations. Any initiated operations in the receiving device are undisturbed. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Clear Lock: The *Clear Lock* command is used by a Commander to cause a Servant device to exit the locked state. When a device receives this command it will set its *Locked** bit. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1

Control Event: The *Control Event* command is used by a Commander to selectively enable the generation of events by a Servant. A one (1) in the enable field enables the generation of the specific event. A zero (0) in the enable field disables the generation of the specific event. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	Enable	Event						

- Event: These bits (6 ← 0) are the identifying bits (14 ← 8) of the event being enabled/disabled. See Section E.4, "Protocol Events", for further information.

The device returns the following data in the Data Low register:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				1	1	1	1	1	1	1	1	1	1	1	0

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The event referenced is not generated by this device.

Control Response: The *Control Response* command is required for devices which implement response signals or response interrupts. It is used to enable response signals or response interrupts on certain response register bit transitions. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	X	B14*	DOR*	DIR*	Err*	RR*	WR*	FHS*

The bits have the following meanings:

- X: Don't care. The value written to this bit has no effect.
- B14*: A zero (0) enables a signal/interrupt on transitions of bit 14 of the Response register. A one (1) disables this capability. Since bit 14 of the Response register is reserved (always one), the value of this bit may be ignored by the Servant.
- DOR*: A zero (0) enables a signal/interrupt on a 0→1 transition of the *DOR* bit. A one (1) disables this capability.
- DIR*: A zero (0) enables a signal/interrupt on a 0→1 transition of the *DIR* bit. A one (1) disables this capability.
- Err*: A zero (0) enables a signal/interrupt on a 1→0 transition of the *Err** bit. A one (1) disables this capability.
- RR*: A zero (0) enables a signal/interrupt on a 0→1 transition of the *Read Ready* bit. A one (1) disables this capability.
- WR*: A zero (0) enables a signal/interrupt on a 0→1 transition of the *Write Ready* bit. A one (1) disables this capability.
- FHS*: A zero (0) enables a signal/interrupt on a 1→0 transition of the *FHS Active** bit. A one (1) disables this capability.

The result data is placed in the Data Low register with the following format. This result is a confirmation/denial of the command.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				1	1	1	1	1	B14*	DOR*	DIR*	Err*	RR*	WR*	FHS*

The bits have the following meanings:

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed as expected.
  - 7_{16}: Command failed - an unsupported bit transition was requested.
- B14*: A zero (0) indicates that interrupt/signal generation on transitions of bit 14 of the Response register is enabled. A one (1) indicates that this capability is disabled. Since bit 14 of the Response register is reserved and should never change, this bit should always be set to one (1).
- DOR*: A zero (0) indicates that interrupt/signal generation on 0→1 transitions of the *DOR* bit is enabled. A one (1) indicates that this capability is disabled.

- DIR*: A zero (0) indicates that interrupt/signal generation on 0→1 transitions of the *DIR* bit is enabled. A one (1) indicates that this capability is disabled.
- Err*: A zero (0) indicates that interrupt/signal generation on 1→0 transitions of the *Err** bit is enabled. A one (1) indicates that this capability is disabled.
- RR*: A zero (0) indicates that interrupt/signal generation on 0→1 transitions of the *Read Ready* bit is enabled. A one (1) indicates that this capability is disabled.
- WR*: A zero (0) indicates that interrupt/signal generation on 0→1 transitions of the *Write Ready* bit is enabled. A one (1) indicates that this capability is disabled.
- FHS*: A zero (0) indicates that interrupt/signal generation on 1→0 transitions of the *FHS Active** bit is enabled. A one (1) indicates that this capability is disabled.

OBSERVATION E.1.3:

A Servant may not implement response signals or response interrupts for some of the defined bits. Such a Servant will always return a one (1) in the corresponding bit position(s) in its response to the *Control Response* command.

End Normal Operation: The *End Normal Operation* command is used to cause a Commander to cease normal operation in an orderly manner. Upon receipt of this command the Commander issues the *End Normal Operation* command to all of its Message Based Servants. The Commander is responsible for halting all its Servants. The "ended" state is defined as follows: All VMEbus Master activity is halted. Pending interrupts are unasserted. No new bus requests or interrupts may be asserted. The device is in a generally inactive state and is ready to accept commands.

The syntax of the *End Normal Operation* command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	1	1	1	1	1	1	1	1	1

When the "ending" operation has completed, response data is placed in the Data Low register in the following format:

Bit #

15 14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
Status	State	Logical Address

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The End Normal Operation command has been successfully executed. The value FE_{16} is reported in the Logical Address field.
  - 7_{16}: The device was already in the CONFIGURE sub-state. The value FE_{16} is reported in the Logical Address field.
  - 6_{16}: The Commander device timed out when waiting for the response from a Servant device. The Servants Logical Address is reported in the Logical Address field.
  - 5_{16}: The device is not able to end its operation in a consistent manner. The value FE_{16} is reported in the Logical Address field.
  - 4_{16}: The Commander device is not able to deactivate a Servant. The Servants Logical Address is reported in the Logical Address field.
  - 3_{16}: An undefined error was caused. The value FE_{16} is reported in the Logical Address field.
- State: This field indicates the state of this device and its sub-tree. It may have the following values:
  - F_{16}: This device and all of its Servant tree are in the CONFIGURE sub-state.
  - B_{16}: This device is in the CONFIGURE sub-state, however at least one device in its Servant tree is in the NORMAL OPERATION sub-state.
  - 9_{16}: This device is in the CONFIGURE sub-state, however the sub-states of the devices in its Servant tree are unknown.
  - 7_{16}: This device is in the NORMAL OPERATION sub-state, however at least one device in its Servant tree is in the CONFIGURE sub-state.
  - 3_{16}: This device and all of its Servant tree are in the NORMAL OPERATION sub-state.
- Logical Address: This field contains the Logical Address corresponding to the status field values.

OBSERVATION E.1.4:

The *End Normal Operation* command, unlike the *Abort Normal Operation* command, allows for the completion of some pending operations before "ending".

Grant Device: The *Grant Device* command is used by the Resource Manager to inform a Commander of the Logical Address of its Servants. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	1	1	1	1	Servant's Logical Address							

Identify Commander: The *Identify Commander* command is used by a Commander to tell a Servant the Commander's Logical Address. This command allows the Servant to reply with signals rather than interrupts. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	1	1	1	0	Commander's Logical Address							

Read Handler Line: The *Read Handler Line* command is used to determine to which VMEbus IRQ line a particular Interrupt Handler in the Servant device is connected. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	1	1	0	0	X					Hand_ID		

- X: Don't care. The value written to this field has no effect.
- Hand_ID: This is a unique identifier of the particular Interrupt Handler being queried. It has a range of 1 to 7. Within a device, multiple Interrupters are identified in consecutive order, starting at 1.

The VMEbus IRQ line number is placed in the Data Low register with the following format.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Status				1	1	1	1	1	1	1	1	1	Line		

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The handler referenced in the Hand_ID field is unknown to this device.
- Line: This is the VMEbus IRQ line number currently assigned. A value of zero (0_{16}) indicates that the handler is disconnected.

Read Handlers: The *Read Handlers* command is used to determine the number of Interrupt Handlers within a Servant device. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1

The number of Interrupt Handlers is placed in the Data Low register with the following format.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	Hand_no		

- Hand_no: This is the number of Interrupt Handlers within this device.

Read Interrupter Line: The *Read Interrupter Line* command is used to determine to which VMEbus IRQ line a particular Interrupter in the Servant device is connected. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	1	1	0	1	X					Int_ID		

- X: Don't care. The value written to this field has no effect.
- Int_ID: This is a unique identifier of the particular Interrupter being queried. It has a range of 1 to 7. Within a device, multiple Interrupters are identified in consecutive order, starting at 1.

The VMEbus IRQ line number is placed in the Data Low register with the following format.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Status				1	1	1	1	1	1	1	1	1	Line		

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The Interrupter referenced in the Int_ID field is unknown to this device.
- Line: This is the VMEbus IRQ line number currently assigned. A value of zero (0_{16}) indicates that the Interrupter is disconnected.

Read Interrupters: The *Read Interrupters* command is used to determine the number of Interrupters within a Servant device. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	1

The number of Interrupters is placed in the Data Low register with the following format.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	Int_no		

- Int_no: This is the number of Interrupters within this device.

Read MODID: The *Read MODID* command is used by a Commander to determine the state of the MODID lines controlled by a given Slot 0 device. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	1	1	0	0	1	1	1	1	1	1	1	1

The resulting MODID status word is placed in the Data Low register with the following format.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	Output Enable	MODID												

- Output Enable: A one (1) in this position indicates that the Slot 0 device is driving the MODID lines.
- MODID: This field contains the state of the 13 MODID lines.

Read Protocol: The *Read Protocol* command is used by a Commander to find out what protocols in addition to the Word Serial protocol that the Servant supports. The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1

The protocol support word is placed in the Data Low register with the following format.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	Device Dependent				RSVD	RG*	EG*	0	PI*	PH*	TRG*	I4*	I*	ELW*	LW*

- Device Dependent: These bits may be defined by the device's manufacturer.
- RSVD: This bit is always one (1).
- RG*: A zero (0) in this position indicates that this device is capable of response generation.
- EG*: A zero (0) in this position indicates that this device is capable of event generation.
- PI*: A zero (0) in this position indicates that this device supports the *Read Interrupters*, *Read Interrupter Line*, and *Assign Interrupter Line* commands.
- PH*: A zero (0) in this position indicates that this device supports the *Read Handlers*, *Read Handler Line*, and *Assign Handler Line* commands.
- TRG*: A zero (0) in this position indicates that this device supports the Word Serial *Trigger* command.
- I4*: A zero (0) in this position indicates that this device supports the VXIbus IEEE-488.2 Instrument protocol.
- I*: A zero (0) in this position indicates that this device supports the VXIbus Instrument protocol.
- ELW*: A zero (0) in this position indicates that this device supports the Extended Longword Serial Protocol.
- LW*: A zero (0) in this position indicates that this device supports the Longword Serial Protocol.

Read Protocol Error: The *Read Protocol Error* command is used by a Commander to tell a Servant device to report its current error state as the 16 bit response to this command. The *Err** bit is always reset before *Write Ready* is asserted in response to this command. After responding to this command, the servant will reset its current error state to "No Error". The syntax of this command is defined in the following table.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1

The error codes are placed in the Data Low register with the following formats.

No error:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Multiple Queries: The device was requested to overwrite previous unread response data.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1

Unsupported Command: This device has received a command that it does not implement.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

DIR Violation: This device has received a command that violates the *DIR* handshake.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1

DOR Violation: This device has received a command that violates the *DOR* handshake.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0

Read Ready Violation: This device has received a command that violates the *Read Ready* handshake.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1

Write Ready Violation: This device has received a command that violates the *Write Ready* handshake.

Bit #															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

All other error codes are reserved for future use by VXIbus.

Read STB: The *Read STB* command is used by a Commander to read the status word from a Servant device. The syntax of this command is defined in the following table.

Bit #															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1

The status byte is placed in the Data Low register with the following format.

Bit #															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	Status Byte							

Read Servant Area: The *Read Servant Area* command is used by the Resource Manager to read the Servant area size of Commanders in the system. The syntax of this command is defined in the following table.

Bit #															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	1	1	1	1	1	1	1	1

The Servant area size is placed in the Data Low register with the following format.

Bit #															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	Servant Area Size							

Release Device: The *Release Device* command is used to cause a Commander to relinquish control of a specified Servant device. After releasing the Servant, the Commander will not attempt any further control of that Servant. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	0	Servant Logical Address							

The command termination status is placed in the Data Low register with the following format.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				1	1	1	1	1	1	1	1	1	1	1	0

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The requested device is not in the Servant list.

Set Lock: The *Set Lock* command is used by a Commander to cause a Servant device to enter the locked state. When a device receives this command it will clear its *Locked** bit. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1

Set Lower MODID: The *Set Lower MODID* command is used by a Commander to change the state of the MODID lines controlled by a given Slot 0 device. This command only effects the state of MODID0 through MODID6 (the lower 7 lines). The enable field, when set to one (1), causes the Slot 0 device to enable its MODID drivers (the function of this field is analogous to the output enable bit in the MODID register of a register based Slot 0 device). The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	0	Enable	MODID6 - MODID0						

Upon the completion of the MODID operation, the device places the following response in the Data Low register:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				1	1	1	1	1	1	1	1	1	1	1	0

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The device is not installed in Slot 0.

Set Upper MODID: The *Set Upper MODID* command is used by a Commander to change the state of the MODID lines controlled by a given Slot 0 device. This command only effects the state of MODID7 through MODID12 (the upper 6 lines). The enable field, when set to one (1), causes the Slot 0 device to enable its MODID drivers (the function of this field is analogous to the output enable bit in the MODID register of a register based Slot 0 device). The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	Enable	X	MODID12 - MODID7					

Upon the completion of the MODID operation, the device places the following response in the Data Low register:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status				1	1	1	1	1	1	1	1	1	1	1	0

- Status: This field indicates the execution status of the command. It may have the following values:
  - F_{16}: The command successfully completed.
  - 7_{16}: Command failed - The device is not installed in Slot 0.

Trigger: The *Trigger* command is used by a Commander to cause a Servant device to trigger any previously armed operation. The syntax of this command is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1

User Defined: The blocks of commands with the following bit patterns are reserved for device specific actions.

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	User Defined Command														

OBSERVATION E.1.5:

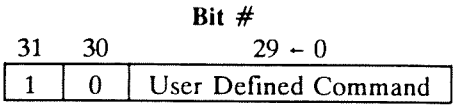
Commands should indicate the presence of an error with a response which has bit 15 cleared to zero (0).

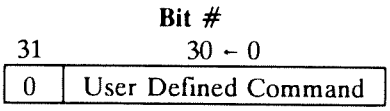
All other commands are reserved for future use by VXIbus.

E.2 LONGWORD SERIAL COMMANDS

The following commands are the standard set of VXIbus device commands which are sent with the longword serial protocol.

User Defined: The blocks of commands with the following bit patterns are reserved for device specific actions.



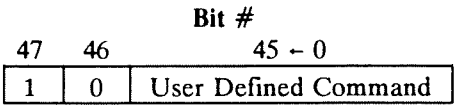


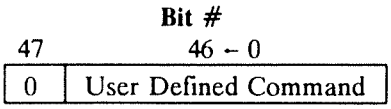
All other commands are reserved for future use by VXIbus.

E.3 EXTENDED LONGWORD SERIAL COMMANDS

The following commands are the standard set of VXIbus device commands which are sent with the Extended Longword Serial protocol.

User Defined: The blocks of commands with the following bit patterns are reserved for device specific actions.





All other commands are reserved for future use by VXIbus.

E.4 PROTOCOL EVENTS

If a Message Based VXIbus Servant interrupts or signals its Commander, it may pass a response or an event in its Status/ID word. If the Status/ID word has bit 15 set to "1" then it is defined as an event. If bit 15 is "0" then it is defined as a response. The lower 8-bits are the Logical Address of the device. This section only covers events. For information on signals see the discussions of interrupts and signals in Section C.2.4, "Message Based Devices".

No Cause Given: This event is sent by a device which has only one reason to signal its Commander. Usually this event will carry the meaning of operation complete. The syntax of this event is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	1	1	1	1	1	1	Sender's Logical Address

Request True This event is sent by a device when the device requires service from its Commander. The syntax of this event is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	1	1	1	1	0	1	Sender's Logical Address

Request False: This event is sent by a device when the device no longer requires service from its Commander. The syntax of this event is defined in the following table.

Bit #

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	1	1	1	1	0	0	Sender's Logical Address

User Defined: This block of events are reserved for use as user defined. The syntax of this event is defined in the following table.

Bit #

15	14	13 12 11 10 9 8	7 6 5 4 3 2 1 0
1	0	User Defined	Sender's Logical Address

All other events are reserved for future use by VXIbus.

F. DYNAMIC CONFIGURATION

Dynamic Configuration is an optional, alternative method of assigning Logical Address to VXIbus devices, based upon slot positions. It allows for each slot to contain one or more devices. Provision is made for the different devices occupying a slot to share address decoding hardware. In addition, Dynamic Configuration requires hardware support in the devices being configured, as well as software support in the Resource Manager.

The following sections present an overview of dynamic configuration, including definition of terms, hardware and software requirements, and a description of the system configuration algorithm.

F.1 DEFINITIONS

In the following pages, the abbreviation DC means "Dynamic Configuration" and the abbreviation SC means "Static Configuration."

- A DC device is a device whose Logical Address can be programmed via the the VMEbus by a DC Resource Manager.
- An SC device is any device whose Logical Address is fixed manually, and cannot be changed, except through manually configuring the device.
- A DC Resource Manager is a Resource Manager which supports Dynamic Configuration.
- A SC Resource Manager is any Resource Manager which does not support Dynamic Configuration, i.e., a Resource Manager as defined in Section C.4.1, "Resource Manager".
- A DC system is a VXIbus system with a DC Resource Manager.
- A SC system is a VXIbus system with a SC Resource Manager.

F.2 DC DEVICE REQUIREMENTS

DC devices have the standard VXIbus configuration registers with some additional requirements. Each DC device has a Logical Address Register, used for setting the device's Logical Address. In addition, the device has certain information encoded in its Offset register at power-on. DC devices are also required to support the MODID line.

F.2.1 Logical Address Register

RULE F.2.1:
Each DC device **SHALL** have a writable Logical Address register located at address offset 0 (00_{16}) of its configuration registers, conforming to the following format:

Bit #	15 ← 8	7 ← 0
Contents	Undefined	Logical Address

The fields of the Logical Address register are defined as follows:

- Undefined: The values written to these bits have no effect.
- Logical Address: The value written to this field is the device's new Logical Address.

RULE F.2.2:
A DC device **SHALL** move its Logical Address before DTACK* is released during the write cycle which transfers the new Logical Address.

RULE F.2.3:
Any DC device which has moved to a new Logical Address **SHALL NOT** respond to any further accesses at its old Logical Address.

OBSERVATION F.2.1:
Like all VXIbus devices, DC devices must have non-volatile Logical Address selectors.

PERMISSION F.2.1:
A Static Configured VXIbus device **MAY** have a functional Logical Address register.

F.2.2 DC Device Logical Address Assignment

A fundamental requirement of Dynamic Configuration is that at power-up time, all DC devices are set to Logical Address 255 (FF_{16}). The Resource Manager will expect to find all DC devices at that Logical Address. Devices at all other addresses are assumed to be SC devices.

RULE F.2.4:
Upon the assertion of SYSRESET*, a DC device **SHALL** assume the Logical Address indicated by its Logical Address Selector.

OBSERVATION F.2.2:
Setting the Logical Address selector of a DC device to 255 (FF_{16}) will cause a DC device to participate in the dynamic Logical Address assignment process. Setting the DC device Logical Address to any other Logical Address will fix the DC device at that Logical Address.

RULE F.2.5:
SC-only devices **SHALL NOT** be set at Logical Address 255 (FF_{16}) in DC systems.

F.2.3 Offset Register

Multiple device modules may share address-decoding hardware. This can result in a significant hardware reduction. In such a case, the devices will share one or more Logical Address bits. A set of such devices is defined to be "address-blocked." These devices will be configured to a block of contiguous Logical Addresses. The Offset register is used to indicate the number of devices sharing the addressing hardware.

RULE F.2.6:
After SYSRESET* is de-asserted, and prior to entering the SELF TEST state, a DC device **SHALL** load its Offset register in the following format:

Bit #	15 - 8	7 - 0
Contents	Undefined	Devices

The fields of the Offset register are defined as follows:

- Undefined: The contents of this field are undefined and not used.
- Devices: The value in this field is the number of devices sharing the common address-decoding hardware. Values of 0, 1, and 255 all indicate a single device.

The devices will have the least significant bits of their Logical Addresses hard wired to predetermined values, and will share the decoding hardware for the most significant bits.

RULE F.2.7:
A set of address-blocked DC devices **SHALL** share address bits in accordance with the following table:

Devices	Shared Bits	Hard-coded bits
2	7 ← 1	0
3 - 4	7 ← 2	1 ← 0
5 - 8	7 ← 3	2 ← 0
9 - 16	7 ← 4	3 ← 0
17 - 32	7 ← 5	4 ← 0
33 - 64	7 ← 6	5 ← 0
65 - 128	7	6 ← 0
129 - 254	-	7 ← 0

RULE F.2.8:
A set of address-blocked DC devices **SHALL** implement the lowest-valued block of contiguous Logical Addresses possible within its hard-coded bits.

OBSERVATION F.2.3:
The above rule guarantees that a set of address-blocked DC devices occupy the lowest-valued addresses available with its block of available addresses.

OBSERVATION F.2.4:
A write to the Logical Address register of an address-blocked module sets the values of all the related devices' Logical Addresses.

F.2.4 MODID Support

A DC device, when at Logical Address 255 (FF_{16}), uses the MODID line as a device select qualifier. The device responds to data transfers when MODID is asserted. After the device has been moved to another address, the device responds at that address independent of the state of the MODID line.

RULE F.2.9:
A DC device **SHALL** use its MODID line as a device select qualifier for Logical Address 255 (FF_{16}).

RULE F.2.10:
A DC device **SHALL NOT** use MODID as a device select qualifier for any Logical Address other than 255 (FF_{16}).

OBSERVATION F.2.5:
The effect of writes to a DC device's Logical Address register is independent of the state of the MODID line. Once the device has been assigned an address other than 255, writes to its Logical Address register will change its logical address, whether or not the MODID line is asserted.

RULE F.2.11:
DC modules which contain multiple DC entities which implement independent address decoding hardware **SHALL** respond to sequential accesses to Logical Address 255 (FF_{16}), qualified with that module's MODID line, one entity at a time, until all entities have been moved to new Logical Addresses.

OBSERVATION F.2.6:
The entities referred to in the preceding rule may be single devices, or a set of devices sharing address-decoding hardware.

RULE F.2.12:
A multiple-board DC module must respond to exactly one MODID line, specified by the device designer.

F.3 DC SYSTEM REQUIREMENTS

The DC system Resource Manager must include the software necessary to implement the three-part configuration algorithm defined in the following section. Additionally, some limitations are imposed on Resource Manager and Slot 0 devices.

RULE F.3.1:
A DC system **SHALL** include a DC Resource Manager.

RULE F.3.2:
DC Resource Managers **SHALL** perform the system configuration responsibilities outlined in Section C.4.1, "Resource Manager", except for the Device Identification and Commander/Servant Hierarchy steps alternatively defined in the following paragraphs.

OBSERVATION F.3.1:
To meet the requirements of Section C.4.1, the DC Resource Manager must reside at Logical Address 0 (00_{16}). Therefore, the Resource Manager must be a SC device and cannot be dynamically configured.

A DC Resource Manager uses the Slot 0 controlled MODID lines as board select qualifiers when assigning Logical Addresses. To access the Slot 0 devices, the Resource Manager must know the Logical Addresses of the system Slot 0 devices.

RULE F.3.3:
Slot 0 devices **SHALL** be SC devices; i.e., Slot 0 devices cannot be dynamically configured.

OBSERVATION F.3.2:
SC devices may be included in a DC system.

RULE F.3.4:
Unless it is the Resource Manager, a VXIbus device **SHALL NOT** modify the logical address of any other VXIbus device by writing to its Logical Address register.

F.3.1 System Configuration Algorithm

To configure a Dynamic Configuration system, the DC system Resource Manager must perform the following three major functions:

- SC Device Identification
- DC Device Logical Address Assignment

  The previous two functions modify the "Device Identification" step of the initialization process outlined in Section C.4.1, "Resource Manager".

- Hierarchy Construction

  This function modifies the "Assign Commander/Servant Hierarchies" step of the initialization process outlined in Section C.4.1, "Resource Manager".

F.3.1.1 SC DEVICE IDENTIFICATION

In any Dynamic Configuration system, there will always be at least one and possibly several SC devices present: the DC Resource Manager and the system Slot-0 device(s). Also, users may wish to install other types of SC devices in DC systems. Since the Logical Addresses of these SC devices are fixed, the DC Resource Manager must first identify all SC devices as specified in Section C.4.1.1, "Device Identification". Having located the SC devices, the DC Resource Manager will not attempt to assign their Logical Addresses to any DC devices.

OBSERVATION F.3.3:
If an SC device is found at Logical Address 255 (FF_{16}), no DC devices may be configured.

F.3.1.2 DC DEVICE LOGICAL ADDRESS ASSIGNMENT

After identifying the SC devices, the DC Resource Manager must assign Logical Addresses to all of the DC devices in the system. This requires that the DC Resource Manager:

- Find every DC device by asserting each MODID line and reading a configuration register at Logical Address 255 (FF_{16}).

  AND

- Write a new Logical Address to each DC device's Logical Address register.

OBSERVATION F.3.4:
No single order of asserting MODID lines is mandated. The order in which a DC Resource Manager asserts the MODID lines is at its manufacturer's discretion. Consequently, the order in which Logical Addresses are assigned to DC devices may vary from system to system.

PERMISSION F.3.1:
In assigning Logical Addresses, the DC Resource Manager may skip the access of slots which are known to contain SC devices.

A DC resource manager may implement any procedure which assigns Logical Addresses to every DC device in the system, such as the following:

For every Slot #1 - 12 in the system do the following steps (1 - 3):

1. Assert the selected slot's MODID line.
2. Repeat the following steps (a - c) until a bus error occurs.
   a. Read the Offset register at Logical Address 255 (FF_{16}).
   b. If the Offset register indicates a single device, then determine the Logical Address to be assigned to that device.

      Otherwise, a set of address blocked devices is being accessed. Determine the Logical Addresses to be assigned to these devices (the lowest address of the block is written in the next step).
   c. Write the new Logical Address to the Logical Address register at Logical Address 255 (FF_{16}).

OBSERVATION F.3.5:
An address blocked group of devices will modify only the shared bits of their Logical Addresses. The hard coded bits will cause the devices to occupy the lowest available Logical Addresses within the block defined by the shared bits. As a consequence, the first device will have Logical Address $B*2^H$, where B is the value assigned to the shared Logical Address bits and H is the number of hard-coded Logical Address bits. The last device will have Logical Address $B*2^H+D$-1, where D is the number of address blocked devices in the group. The Logical Addresses $B*2^H+D$ through

$(B+1)*2^{H}$-1 will be unused and available for assignment to other devices.

RECOMMENDATION F.3.1:
A DC resource manager should verify that each DC device is at its new Logical Address by reading a configuration register at the new Logical Address.

3. De-assert the MODID line.

F.3.1.3 HIERARCHY CONSTRUCTION

DC Resource Managers are not required to respect the system hierarchy configuration encoded in the Servant Area Size of the system devices, as defined in Section C.4.1.4, "Commander/Servant Hierarchies".

RULE F.3.5:
IF the DC Resource Manager does not respect the hierarchy encoded in the Servant Area Size of the system devices,
THEN the DC Resource Manager **SHALL** provide some means for building an initial hierarchy.

G. SHARED MEMORY PROTOCOLS

Shared Memory protocols are intended to supplement the VXI Word Serial protocols. These protocols will allow large amounts of information to be transferred with less overhead than would be required for the Word Serial protocols. In addition, they can be used by devices to communicate outside of the commander/servant relationship. The protocols will be implemented in a layered approach similar to many of the networking protocols used in computer systems.

APPENDIX I. VXIbus REGISTER OVERVIEWS

This section is provided to improve the overview of the VXIbus configuration registers. The following figures contain register maps for the various device types and register contents overviews.

Signature explanation:

[ ]: Optional register or field

: Register with location monitor

( ): Required for particular type of device.

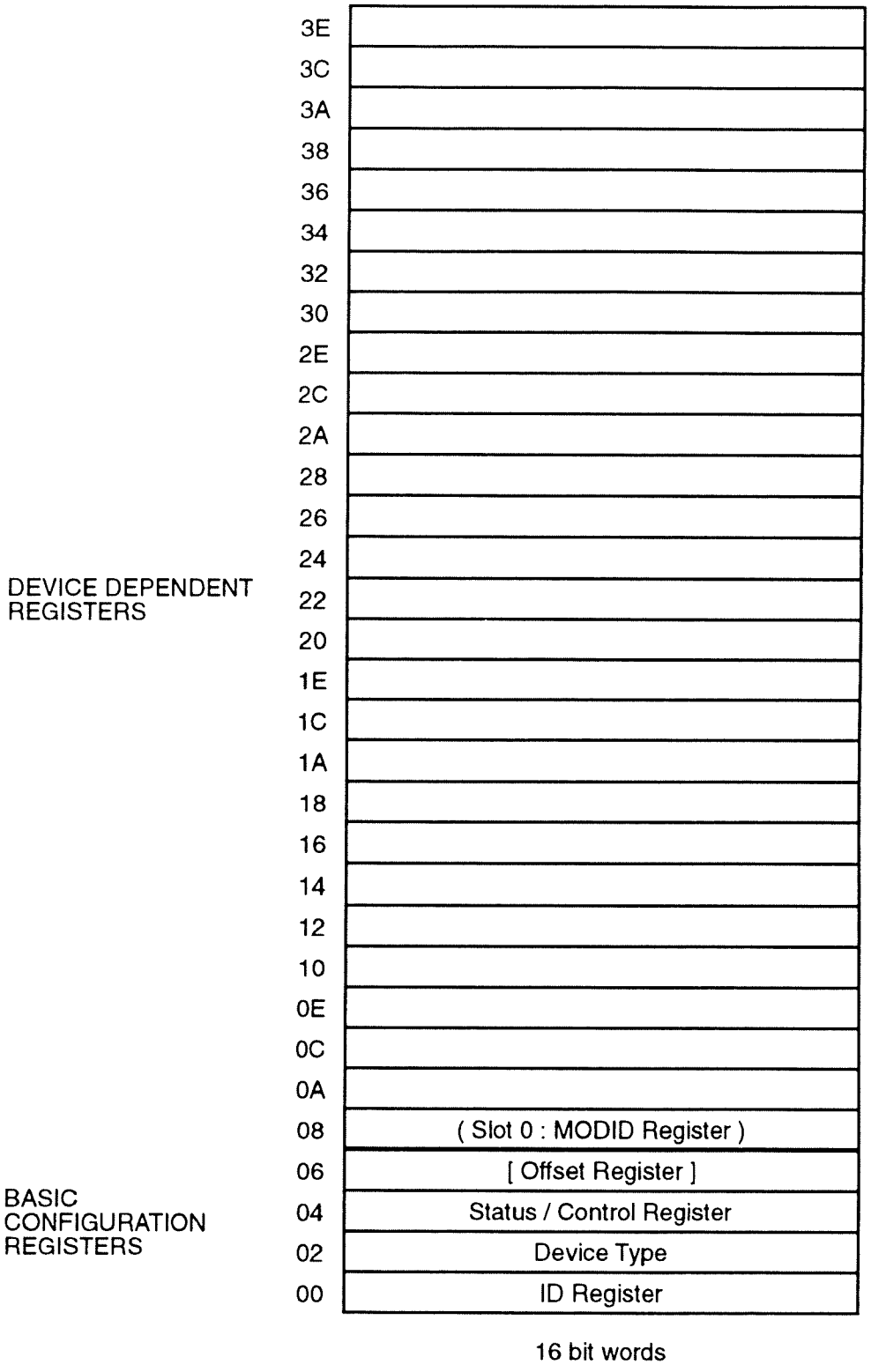


Figure I.1. Register Map for Register Based Devices.

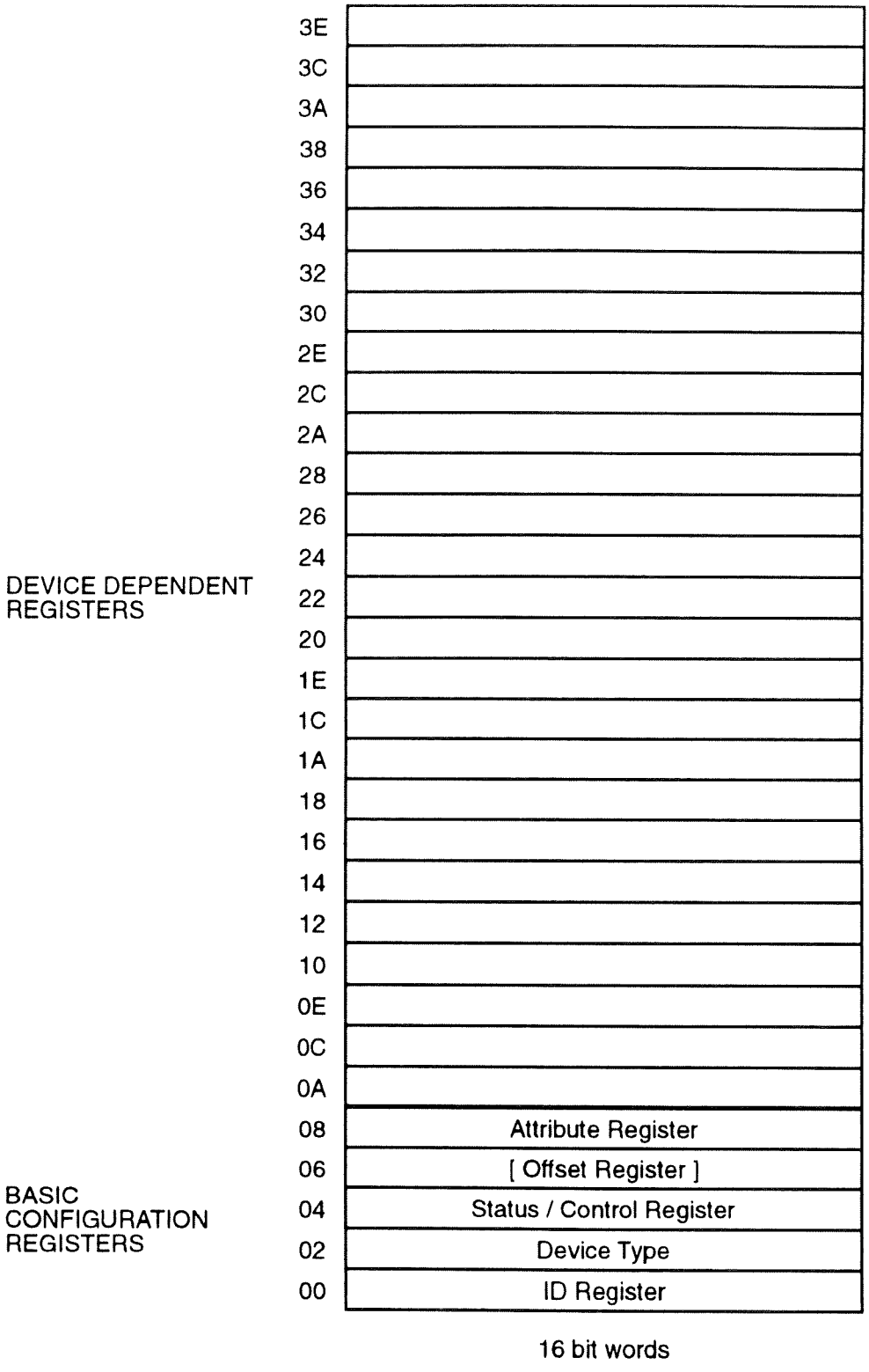


Figure I.2. Register Map for Memory Devices.

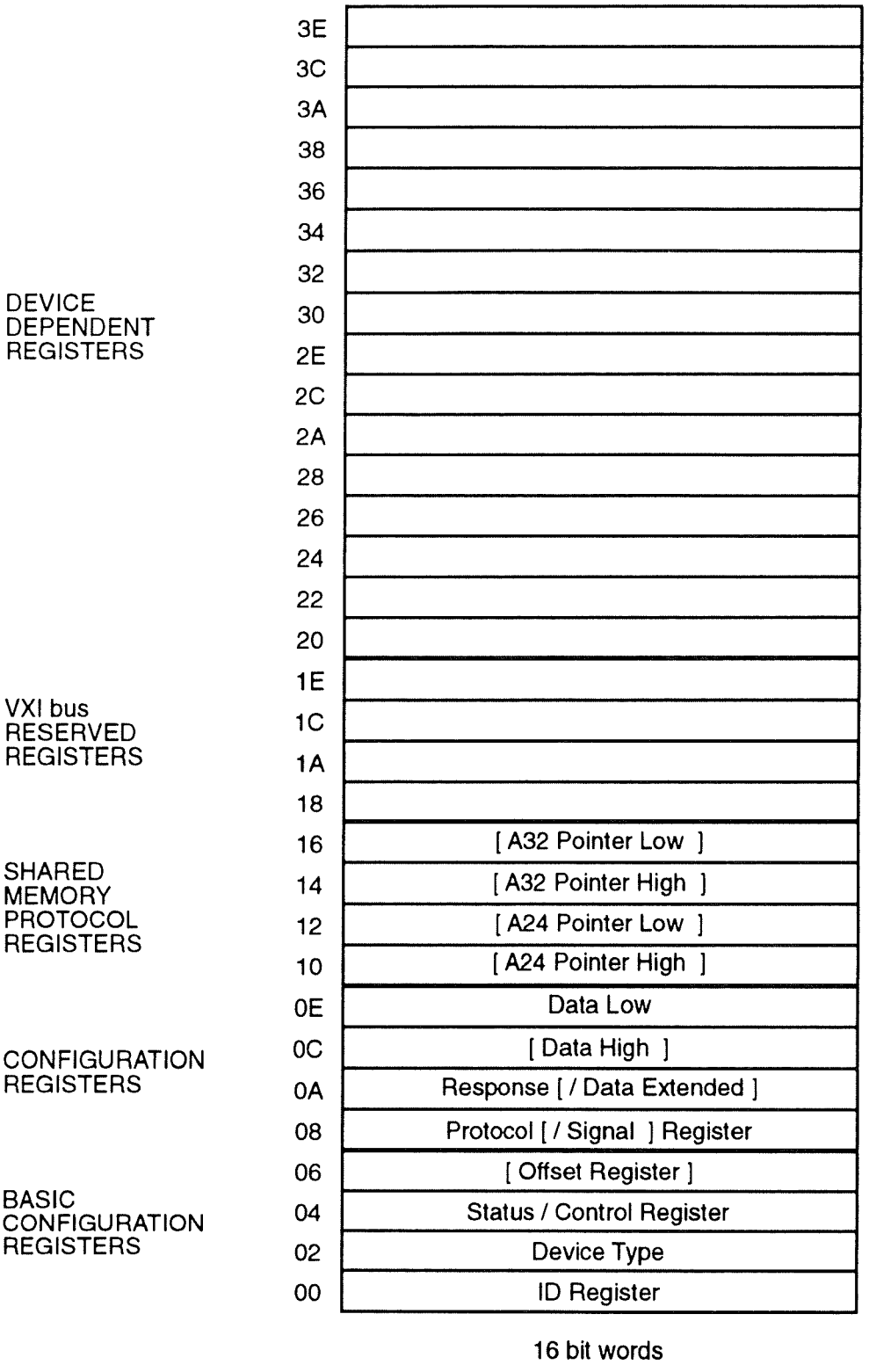


Figure I.3. Register Map for Message Based Devices.

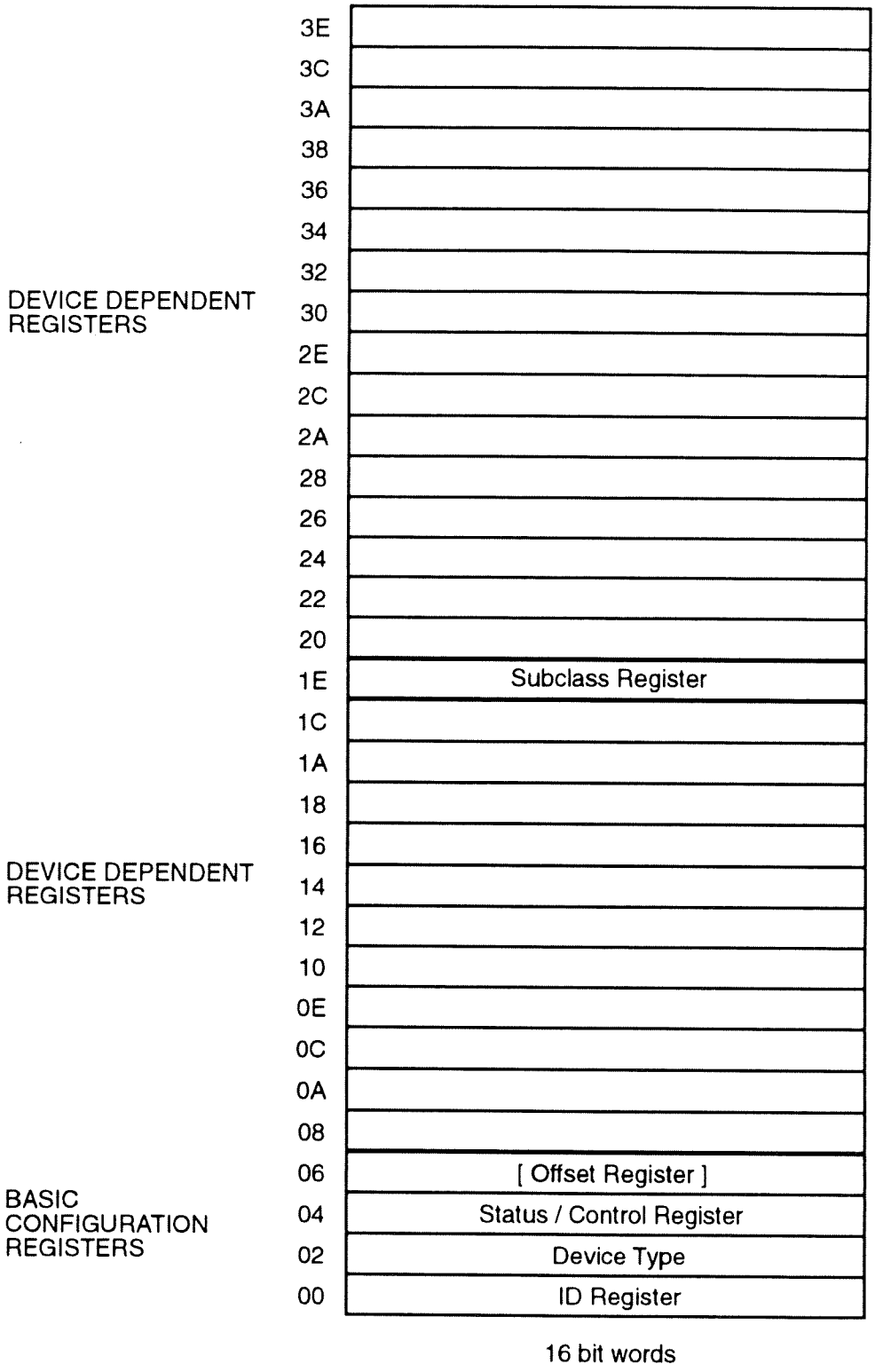


Figure I.4. Register Map for Extended Devices.

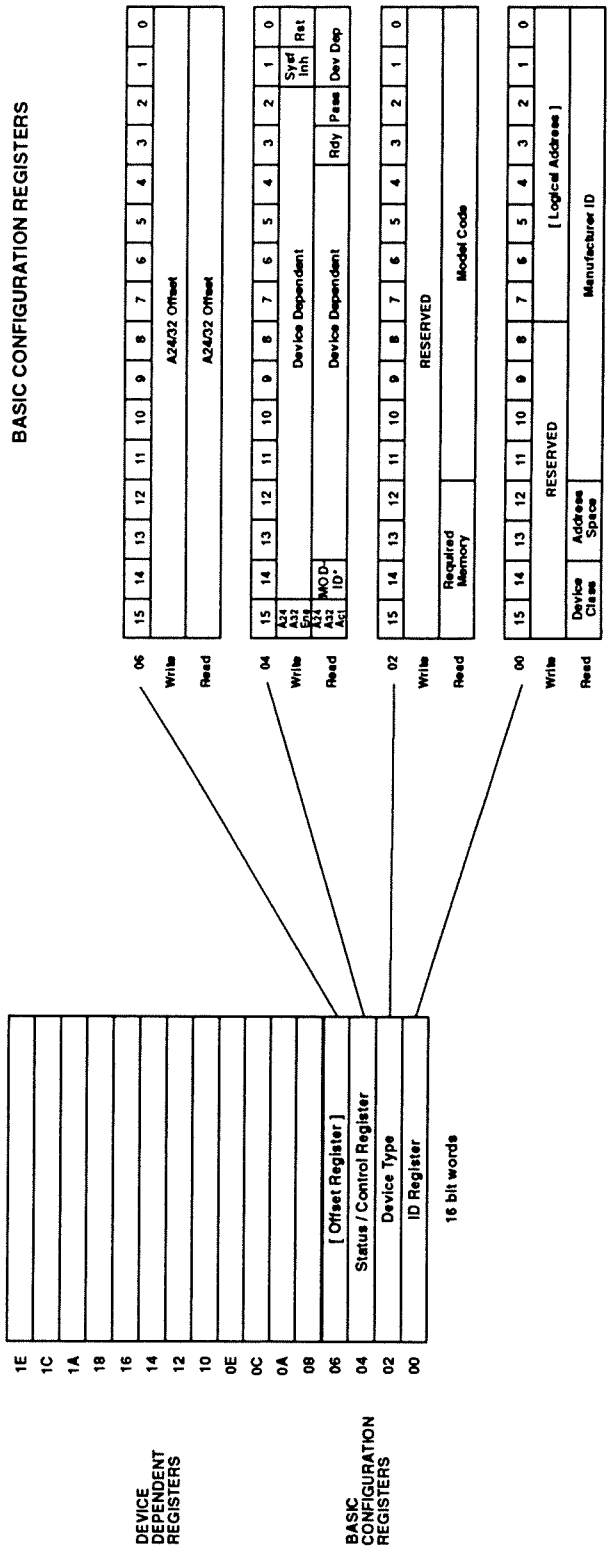


Figure I.5. Register Overview for Basic Configuration Registers.

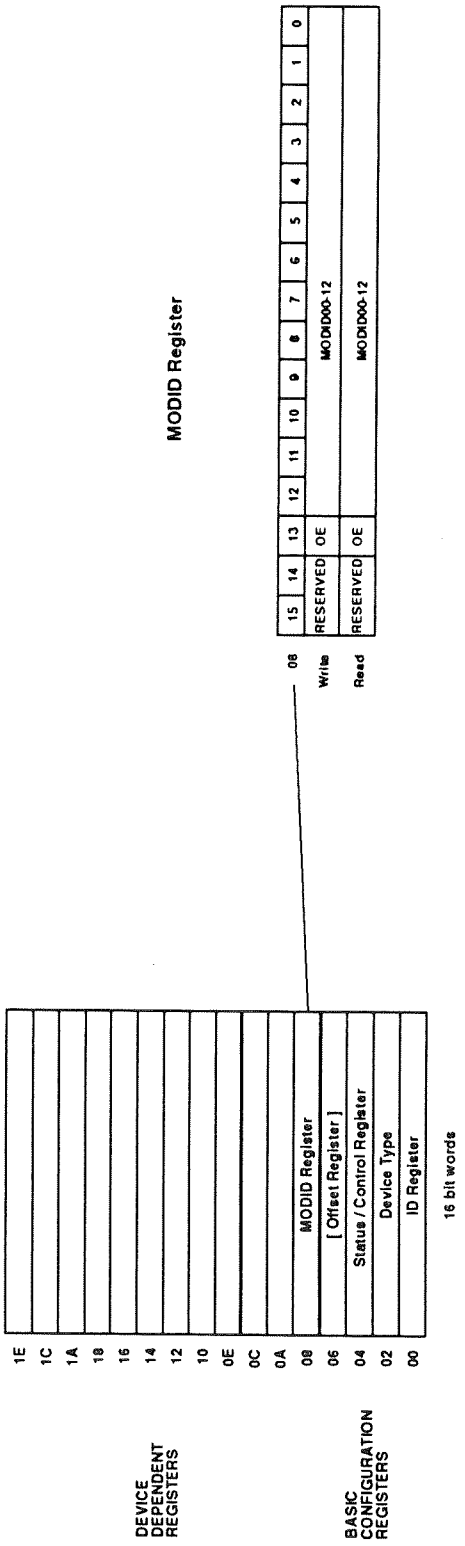

MODID Register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
RESERVED OE MODID00-12
RESERVED OE MODID00-12
08
Write
Read
1E 1C 1A 18 16 14 12 10 0E 0C 0A 08 06 04 02 00
MODID Register
[Offset Register]
Status / Control Register
Device Type
ID Register
16 bit words
DEVICE DEPENDENT REGISTERS
BASIC CONFIGURATION REGISTERS

Figure I.6. Register Overview for MODID Register in Register Based Slot 0 Devices.

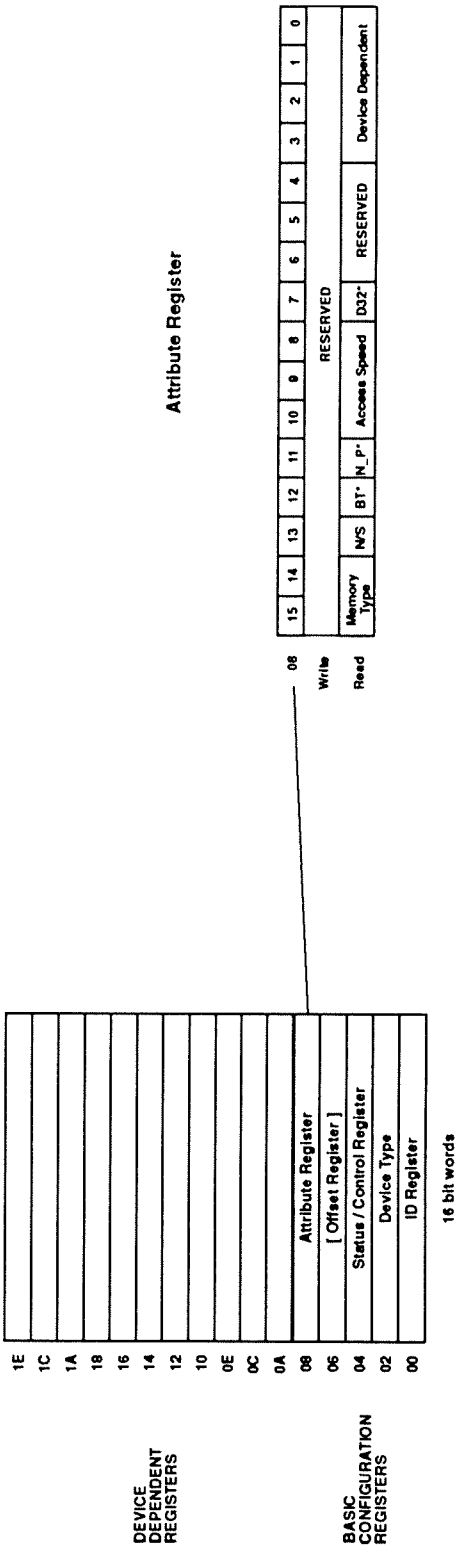


Figure I.7. Register Overview for Attribute Register in Memory Devices.

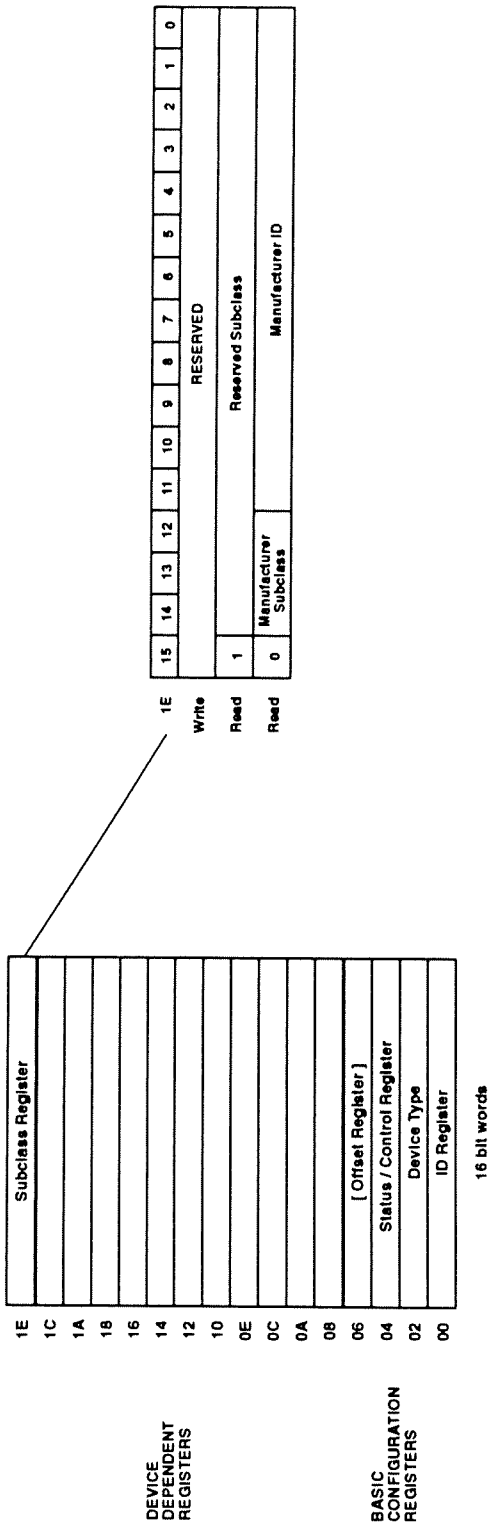


Figure I.8. Register Overview for Subclass Register in Extended Devices.

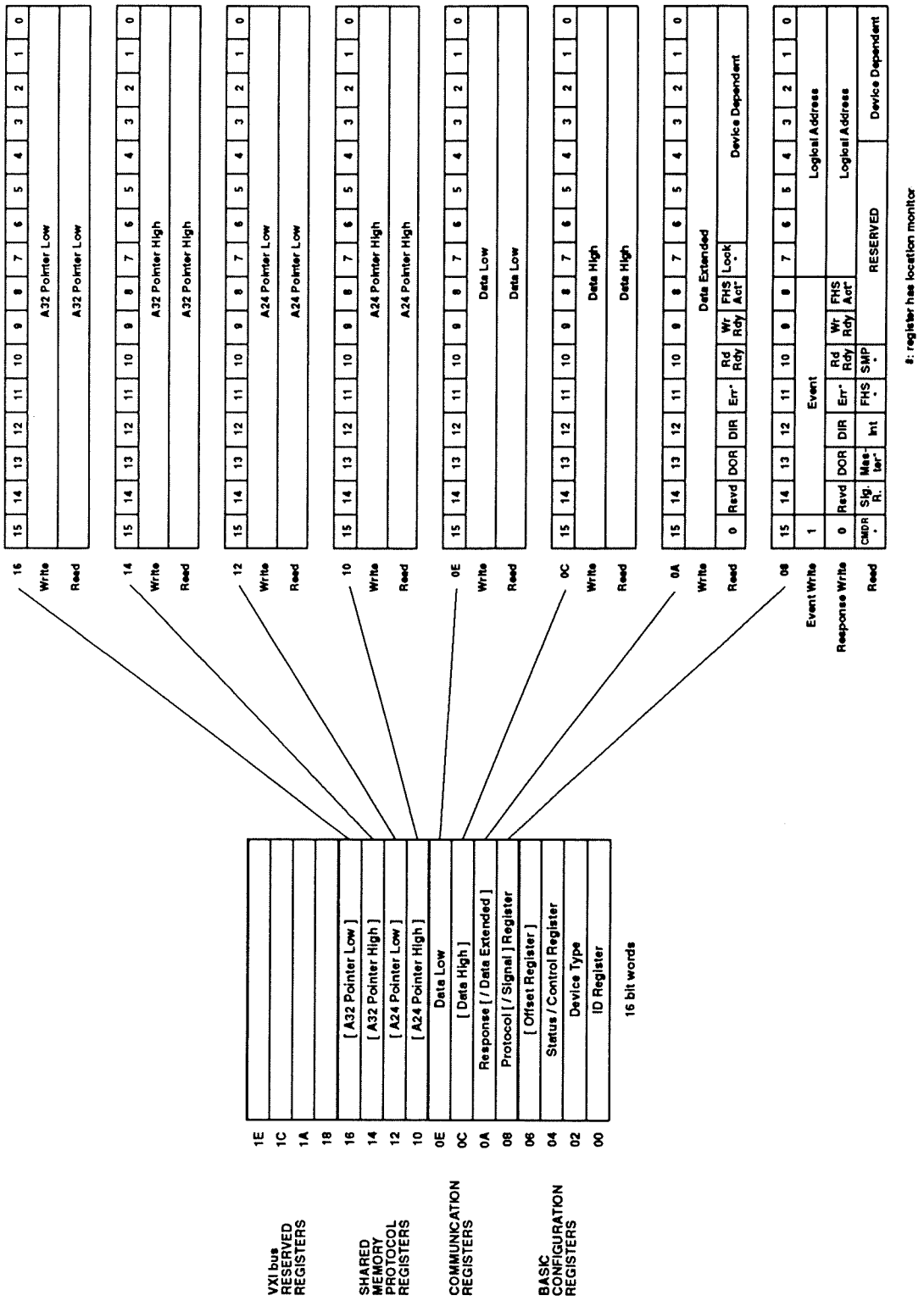


Figure I.9. Register Overview for Communication Registers.

APPENDIX II. SUGGESTED BACKPLANE DESIGN.

II.1 BACKPLANE STRUCTURE

A reasonable backplane layer assignment that minimizes noise and maximizes high frequency performance follows:

TOP (CONNECTOR SIDE)	
1	GROUND PLANE
2	SIGNAL LAYER (TTL)
3	+5 VOLT PLANE
4	GROUND PLANE
5	-5.2 VOLT PLANE
6	-2 VOLT PLANE
7	SIGNAL LAYER (ECL)
8	GROUND PLANE
BOTTOM (SOLDER SIDE)	

By making layers 1 and 8 ground planes, EMI is reduced because of the shielding of the internal traces. Layer 1 also provides the ground layer for attachment of the EMI gasketing to the instrument modules. The thickness between planes 3 and 4, planes 4 and 5, and planes 6 and 7 should be as small as possible (.005 inches is reasonable). This provides high distributed capacitance between those layers and allows high *di/dt* in the voltage planes. One signal layer also exists between the ground layer and the + 5 layer. Since -2 Volts is the signal return for ECL signals, layer 7 should contain as many of the critical ECL signals as possible.

II.2 BACKPLANE LAYOUT

While noise in any logic system can be due to electrostatic coupling, the primary means of coupling is usually due to electromagnetic coupling. The key to designing a low noise system, where electromagnetic coupling is the primary noise source, is the control of current loop areas. Shields or guard stripes do not work. In fact, the main reason for a ground plane is not that is a shield, but that if signal current is adjacent to a neighboring plane, the width of the loop is the spacing between the signal current and that plane. This creates the smallest possible loop area. A designer should always examine the route that any return current may take.

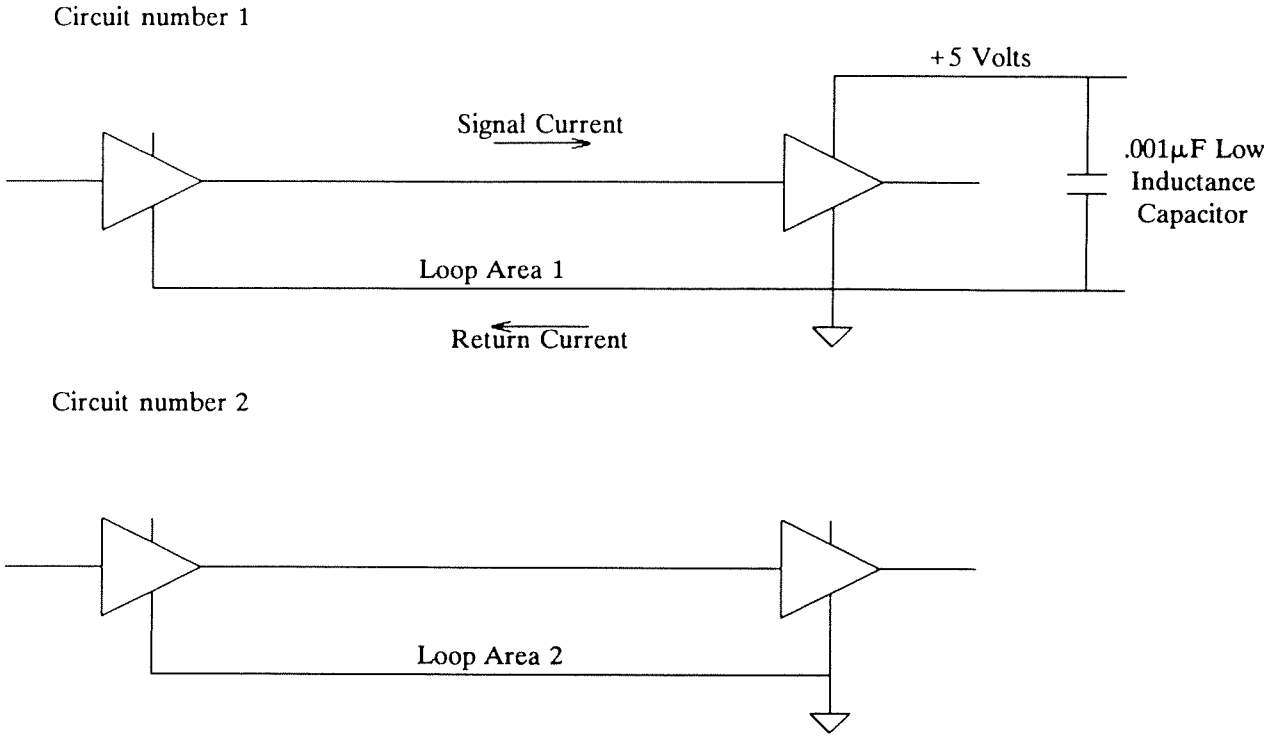


Figure II.1. Typical Logic Circuits.

In Figure II.1, if Circuit number 1 is driven, then there must be a return current. This return current forms a loop, called loop area 1. Circuit number 2 is assumed to be quiet. However there is also a loop area in this circuit which consists of the signal current path and the return current path. The noise induced in the signal line of Circuit number 2 is directly proportional to the loop areas 1 and 2. If these two loop areas are large enough, it is possible to generate a voltage in Circuit number 2 large enough to create a threshold crossing.

Please note that there is another "hidden" loop in Figure II.1. This is the power supply current path. When the receiving gate in Circuit number 1 changes state, a large amount of current is drawn from the supply voltage line. If a voltage plane is used, the loop area due to this current will be small. If a plane isn't used (or if the plane is discontinuous), then the loop area of the power supply return current can be quite large (it could encompass the entire board). If this area is not controlled, it can easily become the dominant noise source. In fact, when ground and voltage planes are not used, this power supply loop is almost always the dominant noise source. If power and ground planes are used, the need for external capacitors is reduced because of the high distributed capacitance.

IT IS HIGHLY RECOMMENDED THAT GROUND AND VOLTAGE PLANES ALWAYS BE USED (PARTICULARLY ON THE BACKPLANE) AND A LOW INDUCTANCE, LOW ESR .001μF CAPACITOR BE PLACED AS CLOSE AS POSSIBLE TO THE POWER AND GROUND PINS OF GATES.

Figure II.2 shows a pair of gates that exist on a PC board with a ground plane, but with a cut made in the ground plane.

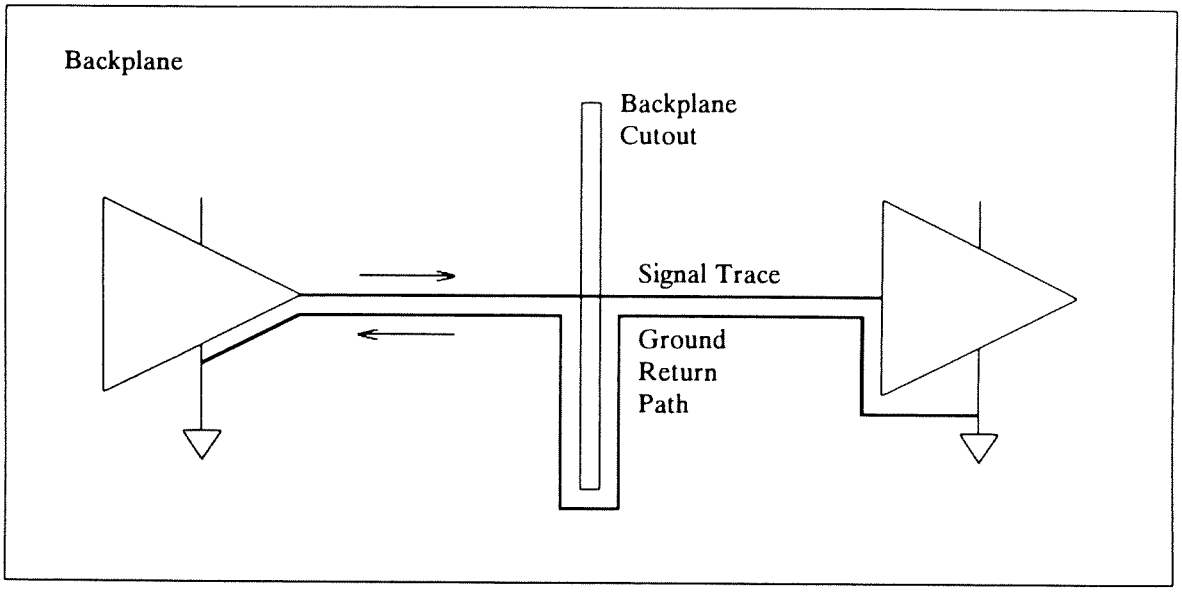


Figure II.2. Loop Area With Cut in Ground Plane.

In Figure II.2 the signal is routed on a layer other than the ground plane. If the "ground plane cut" had not been made the return current would exist adjacent to the signal run, but on the ground plane. With the "cut" the return current is forced away from the signal current, creating a large loop area. The layout shown in Figure II.2 will typically create large amounts of noise. It will also typically receive large amounts of noise.

FOR BEST RESULTS DON'T MAKE ANY CUTS IN A GROUND PLANE OR VOLTAGE PLANE FOR ANY REASON.

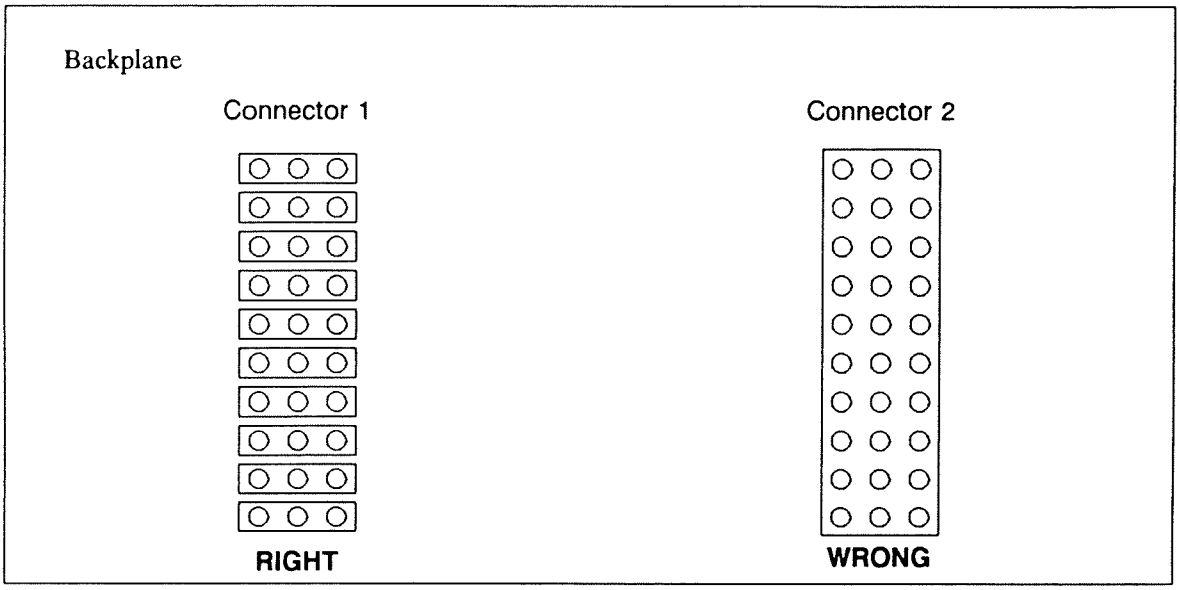


Figure II.3. Backplane Cutouts for Connectors.

In Figure II.3 two connectors are shown. In connector 2, the plane is not routed between pins. This can force large loop areas. In fact, this design is susceptible to simultaneous switching of signal lines, creating *pattern sensitive noise*. This noise can be high enough to cross a threshold for certain bit patterns.

IT IS HIGHLY RECOMMENDED THAT VOLTAGE AND GROUND PLANES BE CONNECTED BETWEEN THE PINS OF ALL BACKPLANE CONNECTORS.

APPENDIX III. Support of Earlier VXIbus Revisions

As the VXIbus specification has evolved, it has been necessary to tighten some of its requirements in order to minimize incompatibilities between devices. This appendix highlights the most important differences between this document and its earlier versions. In addition, it offers recommendations for maximizing compatibility with earlier implementations.

III.1 REVISION 1.2 DEVICES

Some word serial commands that were optional in Revision 1.2 of the VXIbus specification are now required. Revision 1.2 also also permitted some activities that are now prohibited. The following is a minimal set of exceptions to the present specification which, if implemented by the Commander or Resource Manager should provide compatible support of a Revision 1.2 servant-only device. A bit in the response to the *Read Protocol* command identifies Revision 1.2 devices.

RECOMMENDATION III.1.1:

A Resource Manager should implement the following operational changes when communicating with a Revision 1.2 Message Based device.

- Identify any Message Based device that responds to the *Read Protocol* command with bit 15 cleared to (0) as a Revision 1.2 device.
- Identify any Revision 1.2 devices that do not support the *Err** bit. This is indicated by a 1 in bit 7 of the response to the *Read Protocol* command.
- If the device does not support the *Err** bit, the device will report unsupported commands with the "*unrecognized command*" event which was defined for Revision 1.2 Message Based devices. This event will be returned as an event signal if the device is a VMEbus Master, or as an event interrupt if the device is not a Master. The Resource Manager should provide an event handler for these "*unrecognized command*" events and utilize the unsupported command information as necessary.
- A Revision 1.2 device will not return a response to the *Begin Normal Operation* command. Therefore a Resource Manager or Commander should not expect one.
- Revision 1.2 did not prohibit interrupt generation by a device when not in the NORMAL OPERATION sub-state. The Resource Manager should not send the *Read STB*, *Read Protocol Error*, *End Normal Operation*, or *Abort Normal Operation* commands (all optional in Revision 1.2) to a device prior to a *Begin Normal Operation* command to avoid unwanted interrupts prior to the NORMAL OPERATION sub-state.
- The *End Normal Operation* and *Abort Normal Operation* commands were optional commands for servant-only devices in Revision 1.2. Resource Managers or Commanders should either not send these commands to Revision 1.2 devices, or provide for handling of a possible "*unrecognized command*" event interrupt or signal. Since these termination commands may not be supported by the device, proper termination of the device may be application specific.
- The *Asynchronous Mode Control*, *Control Response*, *Control Event*, *Abort Normal Operation*, *End Normal Operation*, and *Read STB* commands, if supported by a Revision 1.2 device did not require bits 15-8 of the "successfully completed" response to these commands to be set to one. The Resource Manager or Commander should mask bits 15-8 when interpreting the response from these commands. The *Control Event* command did not specify a response in Revision 1.2. Therefore, the Resource Manager or Commander should not expect one. The EG* and RG* fields of the *Read Protocol* command were not defined in Revision 1.2. Lack of support for the *Asynchronous Mode Control*, *Control Response*, *Control Event*, *Abort Normal Operation*, *End Normal Operation*, and *Read STB* commands is indicated by an unsupported command error.

- If Programmable Handler (PH) or Programmable Interrupter (PI) capability is supported by a Revision 1.2 device, the responses to these commands differed from the present specification. The *Assign Handler Line* and *Assign Interrupter Line* commands did not permit a response, and the *Read Handlers*, *Read Handler Line*, *Read Interrupters*, and *Read Interrupter Line* commands did not require bits 15-3 of the response to be set to one. The Resource Manager or Commander should not expect a response to the *Assign Handler Line*, or the *Assign Interrupter Line* commands and should mask bits 15-3 when interpreting the response to the *Read Handlers*, *Read Handler Line*, *Read Interrupters*, and *Read Interrupter Line* commands.
- *DOR* and *DIR* support was not required by Revision 1.2 devices. Devices which did not support *DIR* and *DOR* held the associated bits high. A Resource Manager or Commander may not assume that the *DOR* and *DIR* bits in a Revision 1.2 device will accurately reflect the device's state.

III.2 REVISION 1.3 DEVICES

The Word Serial protocol error handling requirements and Fast Handshake protocol requirements for servants have been tightened in Revision 1.4.

Revision 1.3 devices were allowed to queue protocol errors. Queuing is no longer allowed. A device must now un-assert (to 1) its *Err** bit after receiving the *Read protocol error* command, before it re-asserts its *Write Ready* bit. A Revision 1.3 device was allowed to maintain its *Err** indication until it asserts its *Read Ready* bit when writing its last (queued) error code to its Data Low register. This made it difficult for a commander to distinguish between a *Read protocol error* command resulting in an error, a slow response to the command, and a normal response (with more error codes queued). To maintain the best compatibility with such devices, a commander should not test a servant's *Err** bit after a *Read protocol error* command until the servant has asserted its *Read Ready* and *Write Ready* bits, or a very long time has elapsed.

A permission in Revision 1.3 allowed servants in the Fast Handshake mode to continuously assert both their *Read Ready* and *Write Ready* bits. Some Revision 1.3 devices may thus assert their *Read Ready* bits at times when their commanders have not initiated any queries. A commander that implements rigorous error checking might test this bit before sending a word serial query and interpret it as an error condition. In order to avoid this problem, a commander should ignore any *Read Ready* indications from a Fast Handshake servant prior to sending word serial queries.

APPENDIX IV. GLOSSARY OF TERMS

CONFIGURATION REGISTERS: A device's A16 registers which are required for the system configuration process.

BACKPLANE: An assembly, typically a PCB, with 96 pin connectors and signal paths that bus the connector pins. VXIbus systems will have two sets of bussed connectors, called the J1 and J2 backplanes, or have three sets of bussed connectors, called the J1, J2, and J3 backplane.

BOARD: A blank PCB.

BOARD ASSEMBLY: A board and its associated electrical components and connectors.

CHASSIS SHIELD: A shield that resides between two modules and attaches to the mainframe.

COMMAND: Any communication from a commander to a message based servant, consisting of a write to the servant's Data Low register, possibly preceded by a write to the Data High or Data High and Data Extended registers.

COMMANDER : A message based device which is also a bus master and can control one or more servants.

DEVICE: A component of an VXIbus system. Normally, a device will consist of one VXIbus board. However, multiple-slot devices and multiple-device module are permitted. Some examples of devices are computers, multimeters, multiplexers, oscillators, operator interfaces, and counters.

DYNAMIC CONFIGURATION : A method of automatically assigning Logical Addresses to VXIbus devices at system power-on or other configuration times. It allows for each slot to contain one or more devices as well as different devices within a slot to share address decoding hardware.

EVENTS : Signals or interrupts generated by a device to notify another device of an asynchronous event. The contents of events are device dependent.

FAST HANDSHAKE : A high speed mode of operation which uses the same communication registers as the word serial protocol and allows data transfer without the need for polling after each transfer.

EXTENDED DEVICES : A device that has VXIbus configuration registers and a subclass register. This category is intended to allow for definition of additional device types.

EXTENDED LONGWORD SERIAL : A form of WORD SERIAL communication that allows 48 bit data transfers between commanders and servants.

HYBRID DEVICE: A VMEbus compatible device that includes application specific subsets of VXIbus protocols.

LOCATION MONITOR : A function that monitors data transfers over the VMEbus in order to detect accesses to the locations it has been assigned to watch. When an access occurs to one of these assigned locations, the LOCATION MONITOR generates a local signal.

LOGICAL ADDRESS: An 8 bit number which uniquely identifies each VXIbus Device in a system. It defines a device's A16 register addresses, and indicates Commander/Servant relationships.

LONGWORD SERIAL : A form of WORD SERIAL communication that allows 32 bit data transfers between commanders and servants.

MAINFRAME: A rigid framework that provides mechanical support for modules inserted into the backplane, ensuring that connectors mate properly and that adjacent modules do not contact each other. It also provides cooling airflow, and ensures that modules do not disengage from the backplane due to vibration or shock.

MEMORY DEVICE: A device which contains only memory and implements configuration registers.

MESSAGE BASED DEVICE: An intelligent device which implements the defined VXIbus registers and communication protocols.

MODULE: Typically consists of a board assembly and its associated mechanical parts, front panel, optional shields, etc. A module contains everything required to occupy a slot in a mainframe. A module may occupy one or more slots.

OBSERVATION: Observations spell out implications of rules and bring attention to things that might otherwise be overlooked. They also give the rationale behind certain rules, so that the reader understands why the rule must be followed.

OPERATIONAL REGISTER: An operational register is any register on a device that is not required for the system configuration process.

PERMISSION: Permissions are included to clarify the areas of the specification that are not specifically prohibited. Permissions reassure the reader that a certain approach is acceptable, and will cause no problems. The word **MAY** is reserved for indicating permissions.

RECOMMENDATION: Recommendations consist of advice to implementors which will affect the usability of the final device. Discussions of particular hardware to enhance throughput would fall under a recommendation. These should be followed to avoid problems and to obtain optimum performance.

REGISTER BASED DEVICE: A servant only device which supports VXIbus configuration registers. Register based devices are typically controlled by message based devices via device-dependent register reads and writes.

RESOURCE MANAGER : A message based commander located at Logical Address 0 which provides configuration management services such as address map configuration, commander/servant mappings, self test and diagnostics management.

RESPONSES : Signals or interrupts generated by a device to notify another device of an asynchronous event. Responses contain the information in the sender's Response register.

RULE: Rules **MUST** be followed to ensure compatibility for cards in the system. A rule is characterized by the use of the words **MUST** and **MUST NOT**. These words are not used for any other purpose other than stating rules.

SERVANT : A device that is controlled by a commander. There are message based and register based servants.

SERVANT POINTER: An 8 bit number which is the Logical Address of the lowest numbered servant associated with this commander.

SHARED MEMORY PROTOCOL: A communication protocol designed to allow message based devices to communicate by sharing an area of memory accessible to both. The protocol specifies connection and operation sequences to be followed by both devices.

SIGNAL: Any communication between message based devices consisting of a write to a Signal register.

SLOT: A slot is a position where a module can be inserted into an VXIbus backplane. Each slot provides the 96 pin J connectors to interface with the board P connectors. It may have to provide one, two, or three connectors.

SUGGESTION: A suggestion contains advice which is helpful but not vital. The reader is encouraged to consider the advice before discarding it. Suggestions are included to help the novice designer with areas of design that can be problematic.

SYSTEM: A system consists of one or more mainframes that are connected, all sharing a common resource manager. Each device in a system has a unique Logical Address.

VXIbus INSTRUMENT: A VXIbus INSTRUMENT is defined to be a MESSAGE BASED DEVICE which supports the VXIbus Instrument protocols.

VXIbus SUBSYSTEM: A VXIbus Subsystem consists of a central timing module referred to as Slot 0 with up to 12 additional adjacent VXIbus modules. The VXIbus Subsystem bus defines the lines on the P2 and P3 connectors.

WORD SERIAL: The simplest required communication protocol supported by Message Based devices in the VXIbus system. It utilizes the A16 communication registers to transfer data using a simple polling handshake method.

488-VXIbus INTERFACE DEVICE: A 488-VXIbus INTERFACE device is a MESSAGE BASED DEVICE which provides communication between an IEEE 488 Interface and the VXIbus Instruments.

REFERENCES

1. THE VMEbus SPECIFICATION Revision C.1 Micrology pbt

2. ANSI/IEEE Std 488.1-1987 IEEE Standard Digital Interface for Programmable Instrumentation The IEEE, Inc. 345 East 47th St. New York, NY

3. IEEE Std 488.2-1987 Codes, Formats, Protocols, and Common Commands For use with ANSI/IEEE Std. 488.1-1987 IEEE Standard Digital Interface for Programmable Instrumentation The Institute of Electrical and Electronics Engineers, Inc. The IEEE, Inc. 345 East 47th St. New York, NY

INDEX

A

A-size — 45
A16 address space — 5, 97, 99, 101, 122
A16 configuration registers — 101, 120, 158
A16 only device — 103-105
A16 operational registers — 118, 120, 125
A16/A32 device — 104
A24 Pointer Register — 130
A24 RAM — 120, 123
A24 VMEbus — 3, 5, 97, 103, 105, 120
A24/A32 Active — 104, 112, 139
A32 address map configuration — 159
A32 Non-VXIbus — 122
A32 operational registers — 118, 120-121, 125
A32 Pointer Register — 130
A32 REGISTER BASED — 97
A32 VMEbus — 3, 5, 97, 103, 105, 120
Abort Normal Operation — 130, 132-133, 135, 137, 139, 141, 143, 155, 159, 162, 180, 188, 222
Acceptor Handshake — 173
ACFAIL — 9
address blocked devices — 98-99, 101, 106, 118, 120-121, 125, 202-203, 205
address map configuration — 98, 101, 159, 161, 225
address modifiers — 8, 106
address space requirements — 5, 97
address-decoding hardware — 202-203
Addressing Talker — 173
analog summing node — 27
arbiter — 9, 20, 34
arbitration — 3, 8-9, 115, 163
Assign Handler Line — 124, 132, 134, 137, 161, 180, 192, 223
Assign Interrupter Line — 123, 134, 137, 161, 181, 192, 223
ASYNC — 15, 17, 21, 24
asynchronous operation protocol — 222
asynchronous request — 127, 169
Attribute Register — 120
Attributes Register — 121

B

B-size — 45
backplane connectors — 36, 38-39, 41-42, 47, 87, 89, 221, 224-225
backplane ECL lines — 16, 22, 29, 36-37
backplane layout — 218
backplane structure — 218
base address — 99, 101, 105, 118, 125, 159, 164
base address offset — 101, 105, 118, 125, 159
BERR — 8, 127, 149-151, 153-154
Block Transfer Capability — 120
Block Transfer Cycle — 115
BNC — 34
board assembly — 39, 224-225
board connectors — 3-4, 39, 87, 89, 224-225
board injectors/ejectors — 42
board select qualifiers — 204
board warpage lead length — 40
BTO — 8
Buffer — 21, 27
bus devices — 29-31, 109, 115, 128, 154, 158, 166, 173, 180, 188, 224
bus grant — 9, 47, 115
bus instrument control protocols — 166
bus instruments — 4, 30, 166
Bus Request/Grant — 9, 115
bus requesters — 110, 115
bus timer operation — 8
Byte Available — 128, 132, 149, 151-154, 166-167, 169, 173-174, 185
byte blocks — 99, 101
byte request — 5, 128, 132, 149, 151-154, 166-167, 169, 173, 176, 178, 185

C

C-size — 45
center row — 3-4, 10
central handling — 10
chassis ground — 41, 43, 46-49, 83
chassis shield — 4, 43, 48, 224
clear command — 107-108, 135, 139, 143, 151-152, 155, 161, 166-168, 170, 174, 176-177, 185, 196-197, 222
Clear Lock — 132, 166, 168, 173, 177, 185
CLK10 — 10-14, 20, 26, 31-32, 163, 165
CLK100 — 26-27, 31-34, 163
clock signal transmission — 17, 24
CLOSE — 95, 219
close-field — 85-87, 89, 95
CMDR — 126, 134, 159
command field — 104, 180-181, 183-188, 190-191, 195-197, 222
COMMANDER — 224
commander data transfer modes — 146, 148-151
commander servant communication — 98-99, 146, 159, 224
Commander/Servant Hierarchy — 98, 204
commander/servant mapping — 159-161
common system resources — 4, 10, 157, 225
communication ability — 97

communication element – 99, 146
communication protocols – 5-6, 30, 97-99, 122, 125-126, 130, 135, 146-149, 166, 224-226
communication registers – 97-99, 108-109, 117, 124-126, 128, 146-147, 159, 224-226
component height – 40, 47
component lead length – 40
computer systems – 1, 3, 96-99, 160, 207, 224
conducted emissions – 82-83, 94
conducted susceptibility – 83, 95
CONFIGURATION REGISTERS – 224
connection protocol – 30, 225
connector types – 5, 38
control register – 104-105, 107-112, 116, 118, 122, 135, 138, 146, 150, 155-157, 159, 164, 187, 196-197, 225
cooling air – 44, 48
current error state – 155, 193
current flow – 79, 94

D

D sizes – 4
D-size – 35, 45
D-size Slot 0 – 35
D16 STATUS/ID – 113-114, 124
D16 STATUS/ID Transfer – 113
D32 Extension – 114, 117, 124
D32 STATUS/ID – 113, 117
D32 STATUS/ID Transfer – 113
daisy chained bus – 4-5, 29, 115
daisy-chain lines – 9
data byte – 128-130, 151, 166-167, 173-174, 185, 195
Data Extended – 106, 126, 129-130, 143, 147-149, 224
Data High – 126, 128-130, 143, 148-149, 224
Data In Ready – 128
Data Low register – 128-130, 138, 141-142, 148, 150, 153-154, 180-181, 183-188, 190-193, 195-197, 223-224
Data Out Ready – 128
data register – 106, 118, 120, 126-130, 138, 141-143, 146-151, 153-155, 167, 180-181, 183-188, 190-193, 195-197, 223-224, 226
data structures – 147
data transfer, commanders – 128, 146, 148-151, 166-167, 224
data transfer, instruments – 166-167, 174, 176
data transfer bus – 8-9, 115, 148-149, 151
data transfer capabilities – 117, 122
data transfer cycle – 112, 115, 129, 132, 148-151
data transmission – 18, 25
DC device – 201-206
DC device requirements – 201, 204
DC Resource Manager – 201-202, 204-206
DC system requirements – 204
DC system Resource Manager – 201, 204-206
DC voltage specifications – 79
Deassertion of SYSRESET – 158
design rules – 5, 166
determine – 137, 146, 159-161, 167, 190-192, 205
determining – 26, 98-99, 103-104, 118, 121, 125, 135
device addressing – 5, 97-99, 101-103, 105-108, 112, 114, 116-122, 125, 127, 137, 157, 159-161, 164-165, 172, 174, 176, 184, 188, 190, 200-206, 224-225
device capabilities – 98-99, 114, 122, 126, 133, 146, 173
Device Clear – 139, 143, 167, 170, 173
Device Clear Requirements – 167, 170
device communication protocols, – 5, 97-99, 122, 125-126, 130, 135, 146, 148-149, 166, 225-226
device dependent operational registers – 105, 118, 125
device dependent registers – 98, 103-106, 116-119, 121, 125, 138, 145-146, 157, 164
device identification – 10, 158, 204-205
Device Initialization – 104-105, 118, 130-131, 140, 158
device operation – 26, 30, 97, 99, 105, 109-112, 115-119, 127-128, 130, 132-143, 146, 155, 158-162, 168, 176-178, 180, 183-185, 188, 196-197, 200, 222, 225
device slave capabilities – 99, 114
Device Trigger – 173
Device Type Register – 164
DIR – 128, 136, 150-154, 167-168, 170, 174, 177, 179, 187-188, 193, 223
DOR – 128, 136, 150-154, 167, 170, 176, 179, 187, 193, 223
driving rules – 14, 16, 21, 27
DS0 – 121
DS1 – 121
DTB – 115
Dynamic Configuration – 103, 105, 112, 119, 157, 162, 201-202, 204-205
Dynamic Current – 80, 83, 94
Dynamic Module Current – 83

E

ECL signal levels – 29, 37
ECLTRG – 22-27, 34, 36
ECLTRG ASYNC – 24
ECLTRG ESTST – 26, 34

ECLTRG SYNC – 23-24
ECLTRG Trigger Protocols – 34
ECLTRG0-1 lines – 22, 36
electrical specifications – 10
electromagnetic compatibility EMC – 3, 40, 83
electromagnetic coupling – 218
electrostatic coupling – 218
Electrostatic Discharge – 83
EMC – 3, 9, 40-41, 43, 79, 83, 87
emissions – 82-83, 85-87, 89, 94-95
End Normal Operation – 130, 132-133, 135, 137, 139, 141-143, 155, 162, 188, 222
EOI – 174, 176
ERR – 152-153
Error – 128, 133, 143, 154-155, 169, 179, 193, 222
error conditions – 108, 223
error reporting – 128, 154-155, 157, 169, 184, 188, 193
error state – 153, 155, 193
ESTST protocol – 26-27, 34
extended bits – 147, 149, 224
extended device – 4, 26, 29, 99, 106-107, 114, 129-130, 142-145, 148-149, 192, 199, 224
extended device register map – 144
extended devices – 4, 26, 29, 99, 106-107, 114, 129-130, 142-145, 148-149, 192, 199, 224
Extended Longword – 128, 147-149, 192, 199
Extended Longword Serial – 128, 147, 149, 192, 199
extended longword serial commands – 128, 149, 199
extended longword serial protocol – 128, 147, 149, 192, 199
extended longword serial transfer – 147, 149, 224
Extended Register – 114, 129-130, 142
Extended START/STOP – 26
external events – 20, 34
external instruments – 17, 21, 24, 172
external interface – 42, 172, 174, 176
external trigger buffering – 21, 27
external wiring – 42
extraction forces – 42

F

Failed LED – 107-108, 112
fair requester protocol – 115
Fast Handshake Transfers – 126, 129
fast transfers – 126, 129, 149-154, 224
FHS – 126, 129, 136, 139, 149-153, 187-188
filler panels – 41-42
front panel – 39-43, 47, 225
front panel area – 41-43, 47
front panel dimensions – 41
front panel handles – 42
front panel interface – 42
front panel mounting – 41, 47
front view – 38
functional memory locations – 121

G

generalized model – 147
grant device command – 137, 159, 190
ground pins – 43, 83, 95, 219, 221
ground plane – 218-221
ground planes – 218-221
grounding – 14, 27, 36, 41, 43, 46-49, 83, 87, 95, 218-221
Group Execute Trigger – 34, 173
guaranteed setup – 20, 34

H

Handler Line – 124, 132, 134, 137, 161, 179-180, 190, 192, 223
handling – 10, 40, 42-43, 83, 124, 128, 138, 154, 162, 169, 172, 180, 190, 222-223
HARD RESET – 110-111, 118, 140, 155, 159
hierarchical instrumentation system – 146
hierarchical levels – 98, 159-160
hierarchy construction – 204, 206
High Current 3 State – 14
High Register – 129-130
higher level communication protocols – 5, 97
hold times – 15, 18, 20, 25, 27, 34, 131
host computer – 96-98, 160
humidity – 45, 49
hybrid devices – 99, 224

I

IAC – 6
ID – 2, 102-103, 108, 111, 116, 118, 121, 125, 144, 159, 180-181, 190-191
ID Register – 102, 118
IEC 297-3 – 1, 38, 43, 46
IEC publication 603-2 – 38
IEC Publication 68 – 45, 49
IEEE 1014 – 1
IEEE 488 – 1, 4, 166-174, 176-178, 226
IEEE 488 address – 172
IEEE 488 address mapping – 172
IEEE 488 device clear message – 176
IEEE 488 Interface – 166-167, 172-173, 176-178, 226
IEEE 488 remote local – 177

IEEE 488 Serial Poll function — 178
IEEE 488 trigger message — 176
IEEE 488 VXIbus Interface — 172-173, 178, 226
Induced Ripple/Noise — 80, 83, 94
injection/ejection — 42, 48
instrument protocols — 17, 24, 26, 30, 97-98, 166-167, 170-173, 178-179, 192, 226
instruments, commanders — 98, 166-167
interface modules — 3, 5, 16, 42, 96, 99, 224-225
interfacing requirements — 166, 177
interference — 42, 46, 48, 83, 85, 87, 89
intermodule communication — 15
intermodule timing resource — 22
internal state machine — 107
internal traces — 218
interrupt handler capability — 114, 123-124, 137, 160
Interrupt Line — 123-124, 137, 180, 190
Interrupt Request line — 113-114
interrupter capability — 113-114, 123-124, 126, 134, 136-137, 161, 223

L

LBUS — 29-30, 34
level n requester — 115
Load Ripple/Noise — 79-80, 83
Local Bus — 29-30, 36
local bus incompatibilities — 45
local bus lockout keys — 41-42, 45, 49
LOCATION MONITOR — 224
locking devices — 129, 168, 170, 177, 185, 196
logical address 0 — 5, 108, 136, 157, 165, 200-201, 204, 225
logical address 255 — 5, 202-203, 205
Logical Address register — 101, 103, 112, 116, 127, 201-206
logical addresses — 5, 99, 101, 103, 108, 112, 114, 116, 118-119, 125, 127, 136-137, 139, 141, 157, 160-161, 165, 172, 184-185, 188, 190, 195, 200-206, 224-225
longword serial commands — 128, 148-149, 198-199
longword serial protocol — 128, 147-149, 192, 198-199
longword serial transfer — 147-149, 224
longword serial transfers — 147-149, 224

M

mainframe cooling — 3, 48-49, 224
Mainframe Dynamic Current — 80, 94
mainframe output power — 82
mainframe power supply — 9, 49, 79-80, 82, 94
mainframe shielding — 43, 47-48, 86, 224-225
mainframe specifications — 3, 42, 45, 48-50, 82
manufacturer defined subclasses — 144
Manufacturer ID Numbers — 2, 102
Master capability — 114-115, 117, 122-123, 126, 134, 136, 169
Mechanical Specification — 87
memory configuration — 5, 97, 99, 119-121, 224
memory device — 5, 97, 99, 103, 114, 119-122, 126, 146, 159, 207, 224-225
memory device attribute register — 99, 120
memory device operational registers — 120-121
memory device registers — 5, 97, 99, 103, 114, 119-122, 126, 159, 224
Message Based Commander — 98-99, 112, 116, 123-124, 134-135, 137-138, 140-141, 146, 148, 155, 157, 159, 161, 172, 183, 188, 200
Message Based Device Communication Registers — 124-125
message based device registers — 97-99, 106, 114, 122, 124-131, 135, 137-138, 142-143, 146, 148, 151-152, 154, 159, 164, 225-226
message based servant control — 146, 159, 224-225
message based servants — 98-99, 116, 134-135, 137, 140, 142, 146, 150, 155, 157, 159, 161, 183, 188, 200, 224-225
message based Slot 0 device — 164-165
message protocols — 97-99, 122-123, 125-126, 128, 130, 134-135, 137, 143, 146, 148-149, 152-155, 159-161, 166, 169-170, 172, 177, 222, 225-226
messages — 97-99, 104, 106-107, 109, 112-114, 116, 122-143, 146, 148-155, 157, 159-161, 164-167, 169-170, 172-174, 176-179, 183, 185, 188, 200, 222, 224-226
MODID line — 14-15, 104, 163-164, 201, 203-206
MODID register — 164, 192, 196-197, 203, 205
MODID support — 163, 165, 201, 203
MODID00 — 14-15
MODID00-12 — 164
module cooling — 44, 224
Module Current — 83
module environmental — 45, 50
module extraction — 42, 48
module front panels — 40-43, 47, 225
module identification pins — 4, 10
module input — 82-83, 89
module keying — 4, 29-31, 41-42, 45, 49
module power — 3-5, 36, 44-45, 80, 82-83, 94-95
module shielding — 4, 40, 43, 47-48, 86-87, 89, 218, 224-225
module specifications — 1, 3-4, 10, 16, 29, 38-40, 42, 44-45, 50, 82-83, 85, 87
module supply pin — 4-5, 80, 94-95

module width – 4, 40-41
multiple device modules – 5, 34, 202
multiple devices – 5, 34, 123-124, 154, 159, 161, 180-181, 190-191, 193, 202
multiple interrupt handlers – 124, 180, 190
multiple interrupters – 123, 181, 190-191
multiple manufacturers – 4-5

N

NDAC – 176-177
No Cause/Status Given – 114
no data – 3, 128, 151, 155, 223
No Extension Given – 114
No Listen Only Mode – 173
No Request – 115
No Talk Only Mode – 173
noise – 36, 80, 83, 94-95, 218-221
Non-Privileged – 106, 120
Non-VXIbus – 99, 114, 122
Non-VXIbus devices – 99, 122
normal operation – 82-83, 110, 112, 130-143, 149, 155, 157-159, 161-162, 170, 179-180, 183-185, 188, 222
normal transfer mode – 126, 149-151

O

Offset Register – 105, 139, 201-202
open system architecture – 1, 3, 96
Operation Protocol – 162, 222
operation Protocol commands – 149, 162, 222
operation protocol first adds – 130
operational registers – 102, 105, 118, 120-121, 125, 225
optional commands – 129, 132, 148-149, 179, 222
optional dir – 128
optional dor – 128
other device – 4-5, 10, 14, 94, 97-99, 103, 106, 108-109, 112, 115, 118-119, 125, 127, 129, 137-138, 140-143, 146, 157-158, 160-166, 169, 172, 178, 202-206, 224
other devices – 4-5, 10, 14, 94, 97-99, 103, 106, 108-109, 112, 115, 118-119, 125, 127, 129, 137-138, 140-143, 146, 157-158, 160-166, 169, 172, 178, 202-206, 224
other functions – 4, 34, 98, 103, 115, 157, 165, 169
other Slot 0 devices – 4, 10, 14, 163-165
output buffer – 10, 13, 21, 31, 34-35, 143, 167
Output Enable – 164, 192

P

P1 – 3-4, 10, 39, 44, 46, 97, 104, 177
P1 connector – 3-4, 10, 39
P2 – 3-5, 10, 14, 22, 29-31, 36, 39, 49, 104, 163, 168, 226
P2 connector definition – 4
P2 DIN SUMBUS – 29
P2 MODID – 14, 104, 163
P3 connector – 4-5, 31, 39
P3 connector adds – 5, 31
PARD – 80, 94
PASSED – 109, 111-112, 118, 130-131, 135, 139-140, 160
pc board – 39, 220
PCB – 224
Peak – 80, 82-83
Peak Current – 80
Periodic And Random Deviations – 80
Pointer Register – 130
power compatibility – 3, 82
power distribution – 36, 79-80
power distribution network – 36, 80
power management – 82
power monitor – 9, 94
power pins – 4-5, 9, 79-80, 94, 219
power supply – 5, 9, 45, 49, 79-80, 82-83, 94-95, 219
power supply load – 79-80, 94
power-up time – 202
priority interrupt – 9, 112, 117, 122-123, 129
Protocol Events – 125, 186
protocol register – 5, 97-99, 103, 105, 117, 122, 125-126, 128, 130, 134, 137, 143, 146-148, 150-151, 154, 159, 171, 192, 223-226

R

radiated emissions – 85, 87, 89
radiated susceptibility – 87
rated current – 80, 82-83, 94-95
read data registers – 128-130, 147-150, 153, 155, 223
Read Handler Line – 124, 132, 134, 161, 190, 192, 223
Read Handlers – 124, 132, 134, 161, 190, 192, 223
Read Interrupter Line – 123, 133-134, 161, 191-192, 223
Read Interrupters – 123, 133-134, 161, 191-192, 223
Read MODID – 133, 165, 192
read only register – 117, 120, 147, 225
Read Ready – 128-129, 131, 139, 143, 147-151, 153-155, 170, 187-188, 193, 223
Read STB command – 169, 171, 178, 195

ready bits – 2, 102-104, 107-108, 111, 114, 117, 126, 128-129, 131, 137-138, 141-144, 147-155, 164, 167-169, 171, 187-188, 193, 222-223
REC – 101, 123
register based device – 97-99, 106, 114, 117-119, 122, 124-131, 135, 137-138, 142-143, 146, 148, 151-152, 154, 159, 164, 196-197, 225-226
register based device registers – 97-99, 106, 114, 117-119, 122, 124-131, 135, 137-138, 142-143, 146, 148, 151-152, 154, 159, 164, 196-197, 225-226
register protocols – 5, 97-99, 103, 105, 117, 122, 125-126, 128, 130, 134, 137, 143, 146-148, 150-151, 154, 159, 171, 192, 223-226
regulation – 41, 117
Release Device – 132, 134, 137, 195
Release On Acknowledge – 113, 123
Remote/Local – 173
request false – 169, 171, 173, 177-178, 200
request true – 169, 171, 173, 177-178, 200
reserved area – 7, 225
Reserved Registers – 130
resource manager – 5, 98, 105, 112, 115-116, 118, 131, 138, 140-142, 155-162, 165, 190, 195, 201-202, 204-206, 222-223, 225
Response Register – 128, 139
ROAK – 113, 123
ROM – 97, 99, 120-121
RONR – 115

S

SC device – 201, 204-205
SC device identification – 205
SC Resource Manager – 201-202, 204-205
SDC – 176
Segmented Backplane – 10
selected device – 14, 101, 104, 113, 115, 123-124, 127, 160, 172, 203-204
Selective Device Clear – 173
SELF TEST – 108-112, 118, 130, 140-141, 158, 202
self test function – 111
Self Test Operation – 108, 160, 163
SERIAL – 147, 179, 198-199, 224, 226
serial communications – 97-99, 135, 146-149, 157, 166, 224, 226
serial protocols – 97-99, 128, 135, 146-149, 151, 154-155, 162, 166, 169, 176, 178-179, 192, 198-199, 207, 222-224, 226
Servant Area Size – 206
servant data transfer modes – 146, 149-151
Service Request Generation – 178
Set Lock – 132, 166, 168, 173, 177, 196
Set Lower MODID – 133, 165, 196
Set Upper MODID – 133, 165, 197
shared memory protocol word serial commands – 146
Shared System Resources – 98
SHIELD – 224
Signal Register – 126-127
single acceptor protocol – 16-17, 23-24
single line broadcast – 16, 23
single line broadcast trigger – 16, 23
slave only device – 117
Slot 0 CLK100 – 26, 31, 34, 163
Slot 0 device – 14, 163-165, 192, 196-197
Slot 0 functionality – 10, 163-165
Slot 0 MODID – 14-15, 163-165, 192, 196-197, 204
Slot 0 MODIDxx – 14
Slot 0 Module – 10
Slot 0 P2 LBUSA – 30
Slot 0 P2 LBUSC – 30
Slot 0 P3 LBUSA – 30
Slot 0 P3 LBUSC – 30
Slot 0 services – 157, 163-165
Slot 0 SYNC100 – 26, 34-35, 163
SOFT RESET – 109-111, 140, 155, 158-159
SPOLL – 169, 171, 178
SRQ – 169, 171, 177-178
start/stop stst protocol – 15, 20, 26
START/STOP TTLTRG – 20, 26
STARX – 35-36, 163
STARY – 35-36, 163
status register – 103-104, 107-110, 114, 117, 128, 138-139, 141-142, 150, 154, 156, 158, 164, 171, 192, 195
STATUS/ID – 112-114, 117, 124-125, 128
STATUS/ID Transfer – 113
Status/ID word – 124, 200
STST protocol – 20, 26-27
Subclass – 144-145
subclass dependent – 145
Subclass Register location – 145
SYNC100 function – 34
SYNC100 signal – 34, 163
SYSFAIL Inhibit bits – 107
SYSRESET – 9, 16, 22, 104, 107-108, 110-111, 115-116, 118, 140, 158, 202
system configuration algorithm – 201, 204
system devices – 4-5, 10, 97-99, 115, 118-119, 125, 127, 131, 140-141, 146, 157, 159-162, 166, 172, 177-178, 202, 204-207, 224-226
system hierarchy – 96-98, 159, 204, 206
system power-on – 5, 9, 157, 161, 224
system resources – 4-5, 10, 22, 31, 98, 115, 118, 131, 140-141, 155, 157, 159-162, 195, 201, 204-206, 225

System Self Test – 98
system self test management – 158, 161
system wide control hierarchy – 159

T

Talker – 173
top level commander – 137, 157, 159, 161, 183-184
Trigger command – 34, 167-168, 176-177, 192, 197
Trigger function – 34, 168
Trigger operation – 21, 197
Trigger protocol – 34
TTLTRG line – 16-17, 20, 26
TTLTRG loading – 16, 21

U

user defined pins – 3, 10

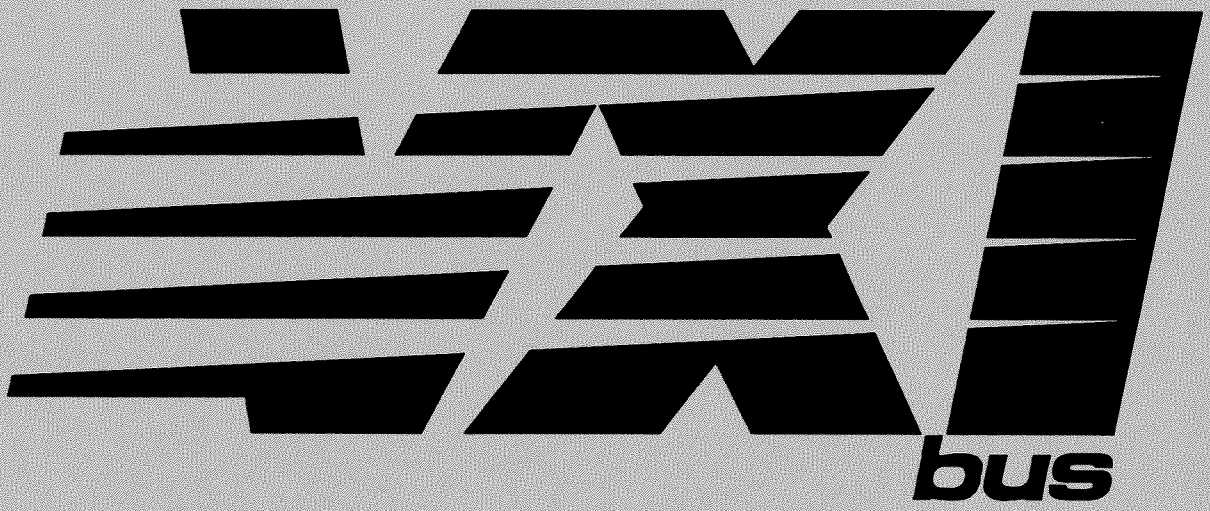
V

valid data – 147
VMEbus Master – 107, 115, 123, 126, 134, 136, 169, 180, 188, 222
VMEbus Specification – 9-10, 16, 36, 110, 115, 239
voltage plane – 218-221
VXIbus Consortium – 2, 102, 238-239
VXIbus Device – 97-99, 102-103, 106, 116-118, 125, 140-141, 144-146, 164-166, 170, 224
VXIbus IEEE-488 – 98, 170, 177, 192
VXIbus Instrument Protocol – 166, 170, 173
VXIbus Interface Device – 98, 166
VXIbus Message Based – 97-99, 106, 114, 116, 122-123, 125, 140-141, 146, 154, 164-166, 170, 200, 226
VXIbus P2 – 3-5, 10, 14, 30-31, 163, 168, 226
VXIbus P3 – 4-5, 30-31, 163, 168, 226
VXIbus Reserved – 130, 144
VXIbus Slot 0 – 4-5, 10, 14-15, 22, 30-31, 46, 103, 157, 163-165, 226
VXIbus Subsystem – 3, 10, 30-31, 103, 163, 226
VXIbus Subsystem P2 – 3, 10, 30, 226
VXIbus Subsystem P3 – 30-31, 226
VXIbus Subsystem Slot 0 – 30, 103, 163, 226
VXIbus's Word Serial Protocol – 97

W

Word Serial Commands – 135-136, 148-149, 165
Word Serial Protocol – 97, 128, 146-147, 162, 169, 192
Write Ready – 129, 131, 147-155, 167, 170, 187-188, 193, 223

VMEbus Extensions for Instrumentation



Extended Register Based Devices and Extended Memory Devices Specification

VXI-2

Revision 1.0

February 7, 1991

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extensions for Instrumentation, Extended Register Based Device and Extended Memory Device Specification VXI-2 Revision 1.0, is authored by the VXIbus Consortium, Inc. and its sponsor members:

A/S Brüel & Kjær
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments, Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce this document by any means for any purpose.

VME Extensions for Instrumentation
Extended Register Based Devices and Extended Memory Devices Specification

A. INTRODUCTION:

The VXIbus System Specification defines a class of devices called Extended Devices (VXIbus System Specification, VXI-1 Rev. 1.3, Section C.2.5). This document is an addendum to the VXIbus System Specification and defines the standard Extended Register Based Device and Extended Memory Device subclasses.

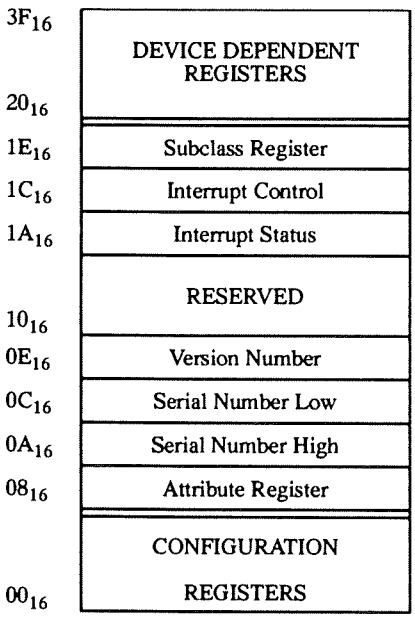
This standard is compatible with:

- VMEbus Extensions for Instrumentation System Specification, Revision 1.3, July 14, 1989. Hereafter referred to as VXI-1 Rev. 1.3.

B. EXTENDED REGISTER BASED DEVICES:

An Extended Register Based Device is an Extended Device equivalent to a Register Based device having an extended set of configuration registers.

The following diagram contains the register map for an Extended Register Based Device:



The Subclass Register and the Attribute Register are required, while the registers from 0A (Hex) to 1D (Hex) are reserved for optional functions defined in this section, or reserved for future VXIbus defined options.

The registers are defined as follows:

Subclass Register: The read contents of this 16 bit register indicates the subclass of this device. The format of the register is as follows:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Contents	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

The bits have the following meanings:

- Bit 15 = 1 indicates that this is a VXIbus defined Extended Device.
- Bits 14-0 = 7FFE (Hex) indicate that this is an Extended Register Based Device.

A write to the Subclass Register has no effect. Its use is reserved for VXIbus definition.

Attribute Register: The read contents of this 16 bit register indicates the implemented options. The format of the register is described as follows:

Bit #	15 ← 3	2	1	0
Contents	RESERVED	IR*	IH*	IS*

The fields are defined in the following way:

(0 indicates option implemented, 1 indicates option not implemented)

- IH : Interrupt Control Register. The device has Interrupt Handler Control capability.
- IR : Interrupt Control Register. The device has Interrupter Control capability.
- IS : Interrupt Status Register. The device has Interrupt Status Reporting capability.
- RESERVED: Reserved for VXIbus definition.

A write to the Attribute Register has no effect. Its use is reserved for VXIbus definition.

Serial Number Registers: The read contents of these two optional 16 bit registers indicates the serial number of the module. The serial number is represented as a 32 bit integer with the 16 least significant bits (LSB) in the Serial Number Low register and the 16 most significant bits (MSB) in the Serial Number High register. The relation between this 32 bit number and actual serial numbers is manufacturer defined. A read value of FFFF(hex) in both registers indicates that this option is not implemented.

A write to the Serial Number Registers has no effect. Their use is reserved for VXIbus definition.

Version Number Register: The read contents of this optional 16 bit register indicates the version number of this module. The format of the register is manufacturer defined. A read value of FFFF(hex) indicates that this option is not implemented.

A write to the Version Number Register has no effect. Its use is reserved for VXIbus definition.

RECOMMENDATION B.1:

Use the following format in the Version Number Register to indicate hardware version number and (if relevant) firmware version number:

Bit #	15 ← 12	11 ← 8	7 ← 4	3 ← 0
Contents	Firmware Main Version #	Firmware Revision #	Hardware Main Version #	Hardware Revision #

- Firmware Main Version # and Revision #:

  These two fields indicate the firmware version number (e.g.: "1.3") encoded as two 4 bit integers.

- Hardware Main Version # and Revision #:

  These two fields indicate the hardware version number encoded as two 4 bit integers.

Interrupt Status Register: The read contents of this optional 16 bit register has the following format:

Bit #	15 ← 8	7 ← 0
Contents	Cause/Status	RESERVED

- Cause/Status: The 8 bits in this field show the Cause/Status field in the STATUS/ID word (bits 15-8), that would be read during an interrupt acknowledge cycle.

RULE B.2:

IF a device has the Interrupt Status Reporting capability

THEN the Interrupt Status Register SHALL be implemented.

OBSERVATION B.3:

The Cause/Status field of the Interrupt Status Register can be used by a Commander to poll an Extended Register Based Device for its current status while interrupt generation is disabled by the Interrupt Mask or IR ENA* in the Interrupt Control Register.

A write to the Interrupt Status Register has no effect. Its use is reserved for VXIbus definition.

Interrupt Control Register: This 16 bit read/write optional register is used to configure VXIbus interrupt lines and control interrupt generation in devices with interrupter capabilities. The format of the register is as follows:

Bit #	15 ← 8	7	6	5 ← 3	2 ← 0
Contents	Interrupt Mask	IR ENA*	IH ENA*	Interrupter IRQ Line*	Handler IRQ Line*

All bits are set to 1 at Hard Reset and Soft Reset.

The fields of the Interrupt Control Register are defined as follows:

- Interrupt Mask: This 8 bit mask is used to enable or disable interrupt generation in a device which has individual causes or status associated with bits 15 to 8 in its STATUS/ID word. For each bit, a zero (0) enables interrupt generation for a cause associated with this bit, while a one (1) disables interrupt generation, but does not clear any pending interrupts. The cause associated with each bit is device dependent.

OBSERVATION B.4:

The mask bits do not have any effect on the current Cause/Status. When a mask bit is changed from one (1) to zero (0), then a pending interrupt cause will generate a VXIbus interrupt.

- IR ENA : A zero (0) in this field enables interrupt generation in the device. A one (1) disables interrupt generation, but does not clear any pending interrupts.

OBSERVATION B.5:

The IR ENA bit has the same effect as the mask bits, but operates on all 8 bits of the Cause/Status.

- IH ENA : A zero (0) in this field enables interrupt handling in the device. A one (1) disables interrupt handling.
- Interrupter IRQ Line : This 3 bit field selects the VXIbus interrupt line associated with the interrupter in the device according to the following table:

Bit # 5	4	3	IRQ Line
0	0	0	IRQ7
0	0	1	IRQ6
0	1	0	IRQ5
0	1	1	IRQ4
1	0	0	IRQ3
1	0	1	IRQ2
1	1	0	IRQ1
1	1	1	Disconnected

- Handler IRQ Line : This 3 bit field selects the VXIbus interrupt line associated with the interrupt handler in the device according to the following table:

Bit # 2	1	0	IRQ Line
0	0	0	IRQ7
0	0	1	IRQ6
0	1	0	IRQ5
0	1	1	IRQ4
1	0	0	IRQ3
1	0	1	IRQ2
1	1	0	IRQ1
1	1	1	Disconnected

A read of the Interrupt Control Register will return the previously written contents in the register.

RULE B.6:

IF a device has the Interrupter Control capability

THEN the IR ENA and the Interrupter IRQ Line fields of the Interrupt Control register SHALL be implemented.

RULE B.7:

IF a device has the Interrupt Handler Control capability

THEN the IH ENA and the Handler IRQ Line fields of the Interrupt Control register SHALL be implemented.

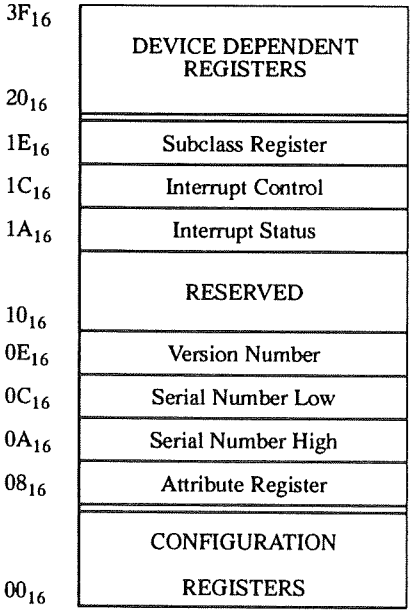
OBSERVATION B.8:

Implementation of the Interrupt Mask field in the Interrupt Control register is device dependent, but is only relevant for devices with Interrupter Control capability.

C. EXTENDED MEMORY DEVICES:

An Extended Memory device is an Extended Device equivalent to a Memory Device having an extended set of configuration registers.

An Extended Memory Device is similar to Extended Register Based Devices with the only exception that the Attribute Register contains special memory attribute fields. The following diagram contains the register map for an Extended Memory Device:



The Subclass Register and the Attribute Register are required, while the registers from 0A (Hex) to 1D (Hex) are reserved for optional functions defined in this section, or reserved for future VXIbus defined options.

The registers are defined as follows:

Subclass Register: The read contents of this 16 bit register indicates the subclass of the device. The format of the register is as follows:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Contents	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1

The bits have the following meanings:

- Bit 15 = 1 indicates that this is a VXIbus defined Extended Device
- Bits 14-0 = 7FFD (Hex) indicates that this is an Extended Memory Device

A write to the Subclass Register has no effect. Its use is reserved for VXIbus definition.

Attribute Register: The read contents of this 16 bit register indicates the memory attributes, and the options implemented. The format of the register is as follows:

Bit #	15 ← 7	6 ← 3	2	1	0
Contents	Memory Attributes	RESERVED	IR*	IH*	IS*

For the definition of the Memory Attributes field refer to the definition of bits 15-7 in the Attribute Register for Memory Devices in Section C.2.3.1.1 of the VXIbus System Specification (VXI-1, Rev. 1.3).

The remaining fields are defined in the following way:

(0 indicates option implemented, 1 indicates option not implemented)

- IH : Interrupt Control Register. Device has Interrupt Handler Control capability.
- IR : Interrupt Control Register. Device has Interrupter Control capability.
- IS : Interrupt Status Register.
- RESERVED : Reserved for VXIbus definition.

A write to the Attribute Register has no effect. Its use is reserved for VXIbus definition.

For the definition of Serial Number Registers, Version Number Register, Interrupt Status Register and Interrupt Control Register, refer to Section B in this document.

D. RESOURCE MANAGEMENT:

Extended Register Based Devices and Extended Memory Devices can have their IRQ lines allocated by the Resource Manager or their Commander.

Because Extended Devices must implement the standard set of VXIbus configuration registers, they can have their Logical Address and Offset registers configured by a normal Resource Manager that is otherwise unable to utilize the added capabilities of Extended Register Based Devices and Extended Memory Devices.

OBSERVATION D.1:

If a Resource Manager is unable to utilize the added capabilities of Extended Register Based Devices and Extended Memory Devices then an Extended Register Based Device or Extended Memory Device will be configured as a Register Based Device.

In a system with a normal Resource Manager, a Commander must allocate IRQ lines to Extended Register Based Device and Extended Memory Device servants as defined in Rule C.2.81 of the VXIbus System Specification (VXI-1, Rev. 1.3). The Commander should first check that the IRQ lines have not already been allocated. This can be done by reading the Interrupter IRQ Line field and the Interrupt Handler IRQ Line field of the Interrupt Control registers. If the values of these fields are different from 7(hex), then the IRQ lines have already been allocated.

A Resource Manager that can configure Extended Register Based Devices and Extended Memory Devices in addition to normal Resource Manager functions has the Extended IRQ Line Allocation capability. A Resource Manager with this capability is responsible for the allocation of the IRQ lines for Extended Register Based Devices and Extended Memory Devices that have Interrupter Control or Interrupt Handler Control capability. The devices are configured using their Interrupt Control registers.

E. REGISTER OVERVIEWS:

The following figures contain register content overviews for Extended Devices:

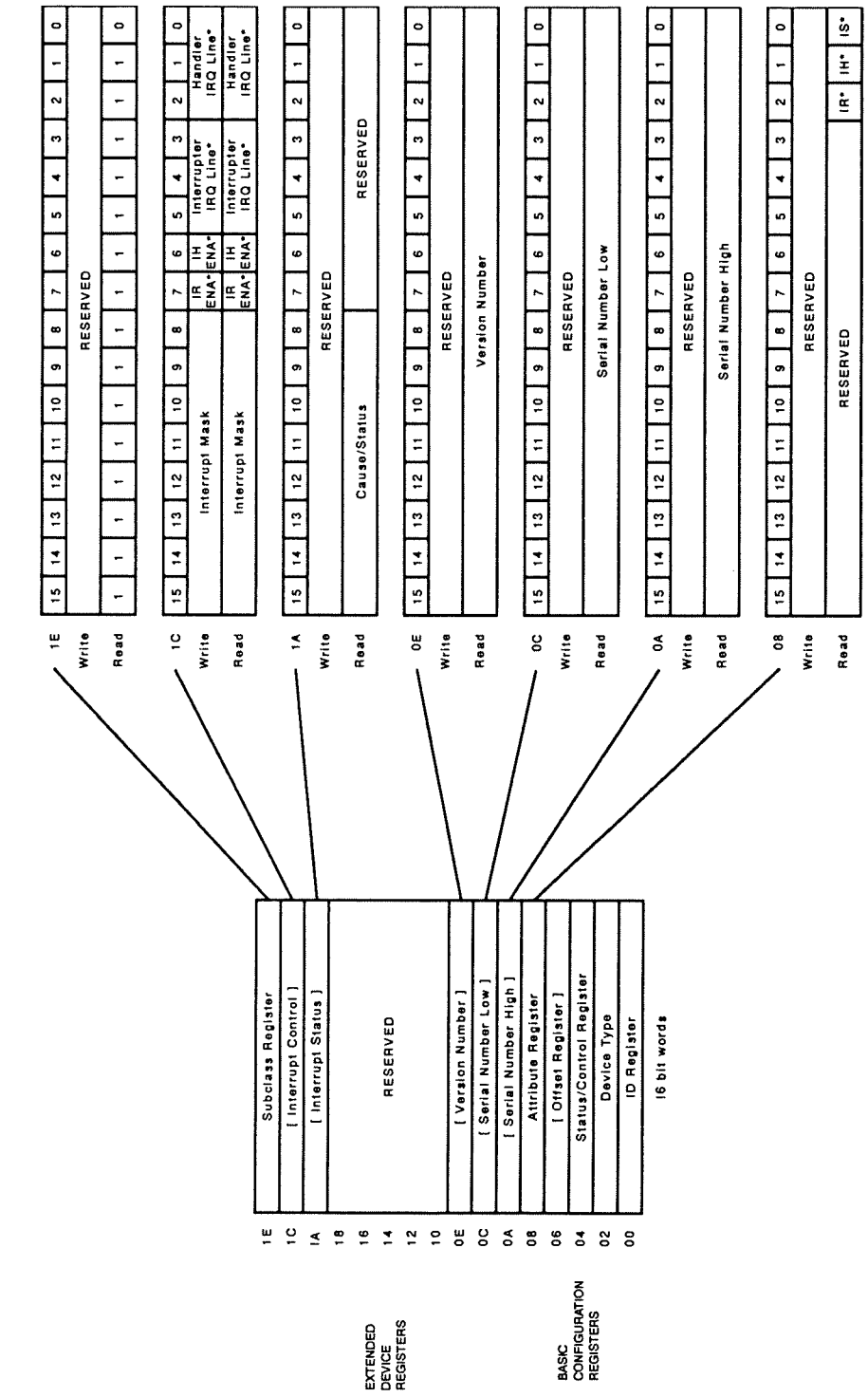


Figure 1. Register overview for Extended Register Based Device Registers.

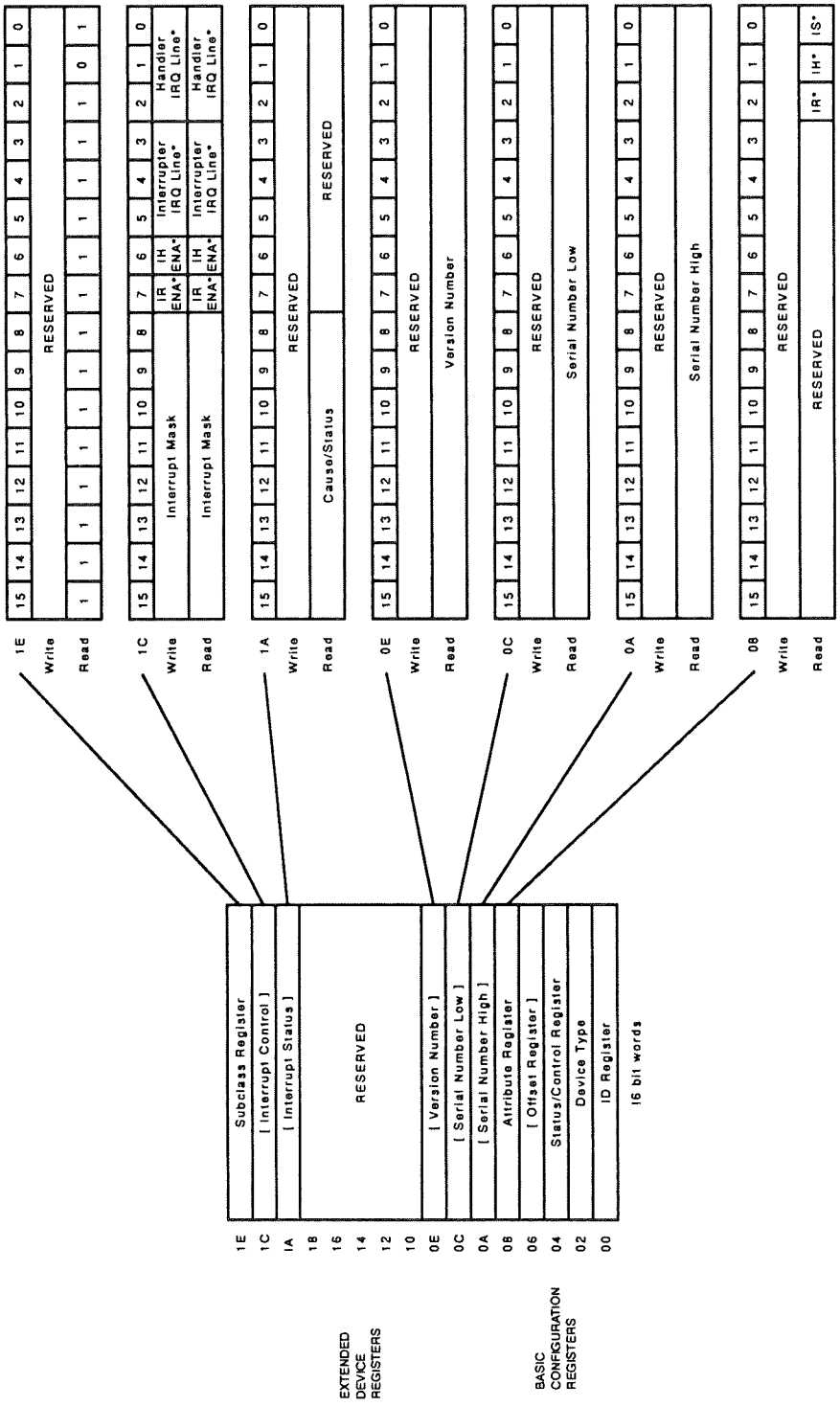
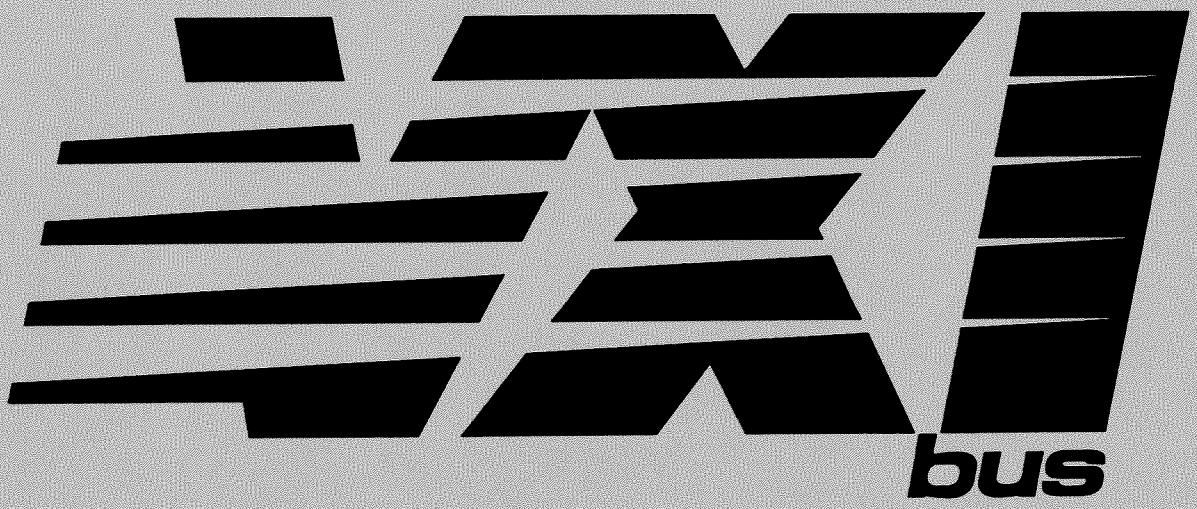


Figure 2. Register overview for Extended Memory Device Registers.

VMEbus Extensions for Instrumentation



Word Serial Commands for Version and Serial Number Identification Specification

VXI-3

Revision 1.0

February 7, 1991

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extensions for Instrumentation, Word Serial Commands for Version and Serial Number Identification Specification VXI-3, Revision 1.0, is authored by the VXIbus Consortium, Inc. and its sponsor members:

A/S Brüel & Kjær
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments, Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce this document by any means for any purpose.

VME Extensions for Instrumentation
Word Serial Commands for Version and Serial Number Identification Specification

A. INTRODUCTION:

The commands specified in this document are an addendum to the VXIbus System Specification and are optional commands for device identification by serial number and version number.

This standard is compatible with:

- VMEbus Extensions for Instrumentation System Specification, Revision 1.3, July 14, 1989. Hereafter referred to as VXI-1 Rev. 1.3.

B. COMMAND SPECIFICATION:

The following commands, that are defined in this document, are optional for all Message Based Devices:

- Read Serial Number High
- Read Serial Number Low
- Read Version Number

These commands are all sub-state independent as defined in the VXIbus System Specification (VXI-1 Rev. 1.3) Section C.2.4.4.1.

Read Serial Number High: The Read Serial Number High command is used to determine the 16 most significant bits of the 32 bit integer, that indicates the serial number of a device. The syntax of the command is defined in the following table:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	1	0	1	1	1	1	1	1	1	1

The response is placed in the Data Low Register which has the following format:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Serial Number High															

Read Serial Number Low: The Read Serial Number Low command is used to determine the 16 least significant bits of the 32 bit integer, that indicates the serial number of a device. The syntax of the command is defined in the following table:

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1

The response is placed in the Data Low Register which has the following format:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Serial Number Low															

OBSERVATION B.1:

The relation between the 32 bit integer read with the Read Serial Number commands and the actual serial number is manufacturer defined.

Read Version Number: The Read Version Number command is used to determine the version number of a device. The syntax of the command is defined in the following table:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1	1

The response is placed in the Data Low Register which has the following format:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Version Number															

OBSERVATION B.2:

The format of the version number is manufacturer defined.

RECOMMENDATION B.3:

Use the following format of the Version Number to indicate hardware version number and (if relevant) firmware version number:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Firmware Main Version #				Firmware Revision #				Hardware Main Version #				Hardware Revision #			

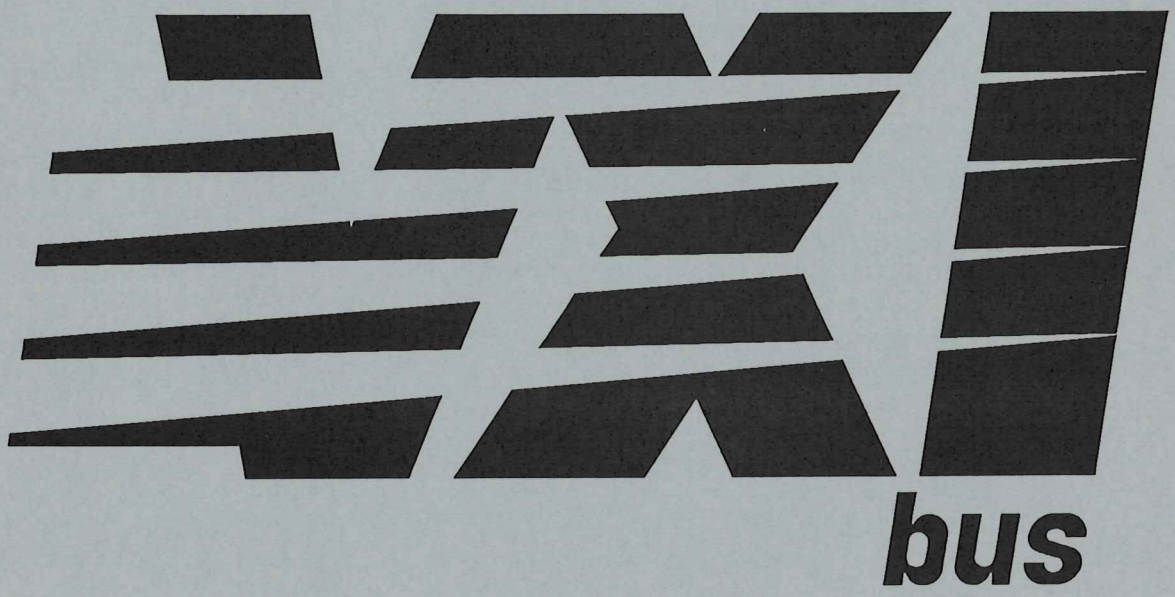
- Firmware Main Version # and Revision #:

  These two fields indicate the firmware version number (e.g. "1.3") encoded as two 4 bit integers.

- Hardware Main Version # and Revision #:

  These two fields indicate the hardware version number encoded as two 4 bit integers.

VMEbus Extensions for Instrumentation:



Common Mnemonics

Specification

VXI-4

REVISION 1.0

June 23, 1991

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extensions for Instrumentation: Common Mnemonics VXI-4 Revision 1.0 is authored by the VXIbus Consortium, Inc. and its sponsor members:

A/S Bruel & Kjaer
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments, Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce and distribute this document by any means for any purpose.

CONTENTS

A. INTRODUCTION 1
B. OVERVIEW 2
C. ABBREVIATION RULES 3
D. ABBREVIATIONS 4

VMEbus Extensions for Instrumentation:

Common Mnemonics

A. INTRODUCTION

The VXIbus architecture defines several common terms. Many of these are rather long phrases, awkward to use in VXIbus related discussions. In order to ease this difficulty, a set of abbreviations, or mnemonics, for many VXIbus terms has been developed. This specification lists these mnemonics, along with a set of abbreviation rules used in their development.

This specification is compatible with:

- VMEbus Extensions for Instrumentation System Specification, Revision 1.3, July 14, 1989, hereafter referred to as VXI-1 Rev 1.3.
- The VMEbus Specification, Revision C.1 (aka IEEE 1014-1987), October, 1985.
- IEEE Std 488.2-1987 Codes, Formats, Protocols, and Common Commands for Use with ANSI/IEEE Std 488.1-1987 IEEE standard Digital Interface for Programmable Instrumentation. The IEEE, Inc.
- Standard Commands for Programmable Insrumentation (SCPI), Version 1991.0, April 1991. SCPI Consortium.

B. OVERVIEW

The goal of this specification is to define a set of rules by which mnemonics referring to VXIbus resources or protocols can be derived and to keep these rules consistent with IEEE 488.2 and VXI 1.3.

The mnemonics are abbreviations for VXIbus terms, and can be used by any language or command syntax. Shortform mnemonics are indicated by CAPital- ization. Longform mnemonics are the combination of upper and lower case letters.

The mnemonics are derived by taking the corresponding VME or VXI term, as spelled in the corresponding specification, and applying abbreviation rules to reduce the number of characters to four.

C. ABBREVIATION RULES

1. The shortform mnemonic is the first four characters of the term unless:
    1. the fourth character is a vowel, then the mnemonic is the first three characters of the term. This rule is not used if the term has exactly four characters.
    2. the longform mnemonic is defined from a phrase rather than a single term, then the longform mnemonic is the first character of each term except that the entire last term is spelled out. The shortform mnemonic is derived using the above rules on this longform mnemonic.
    3. the mnemonic is an industry standard term or has previously been defined by VME or VXI, then that definition takes precedence.
2. Any numeric suffix is added after the truncation rules (1.) are applied.
3. All terms must have both a shortform and longform specified.

D. ABBREVIATIONS

When referring to the following common VXIbus resources or protocols, the following standard mnemonics are used:

VXIbus term	Standard Mnemonic
A24 Pointer	A24Pointer
A32 Pointer	A32Pointer
Abort Normal Operation	ANO
Acceptor Handshake	AHANdshake
ACFAIL	ACFAIL
Acknowledge	ACKnowledge
Address Modifiers	AMODifiers
Analog Summing Bus	ASBus
Analog Summing Node	ASNode
Assign Handler Line	AHLine
Assign Interrupter Line	AILine
Asynchronous (Trigger Protocol)	ASYNC
Asynchronous Mode Control	AMControl
Attribute	ATTRibute
Base Address	BADDress
Begin Normal Operation	BNO
Block Transfer	BTRansfer
Bus Error	BERR
Bus Grant n*	BGrant*n*
Bus Request n*	BRQ*n*
Bus Timeout	BTO
Byte Available	BAVailable
Byte Request command	BRQ
Byte Transfer Protocol	BTPRotocol
Clear Command	CLR
Clear Lock	CLOCk
CLK10 line	CLK10
CLK100 line	CLK100
Commander	CMDR
Commander/Servant Hierarchy	CSHierachy
Conducted Emissions	CEMISsions
Conducted Susceptibility	CSUSceptibility
CONFIG REG INIT	CRINit
CONFIGURE Sub-State	CONFigure
Control Event	CEVent
Control	CONTrol
Control Response	CRESponse
Data Byte	DAB
Data Extended	DEXTended
Data High	DHIGh
Data In Ready	DIR
Data Low	DLOW
Data Out Ready	DOR
Data Transfer Bus	DTB
Device Clear	DCLear
Device Type	DTYPe

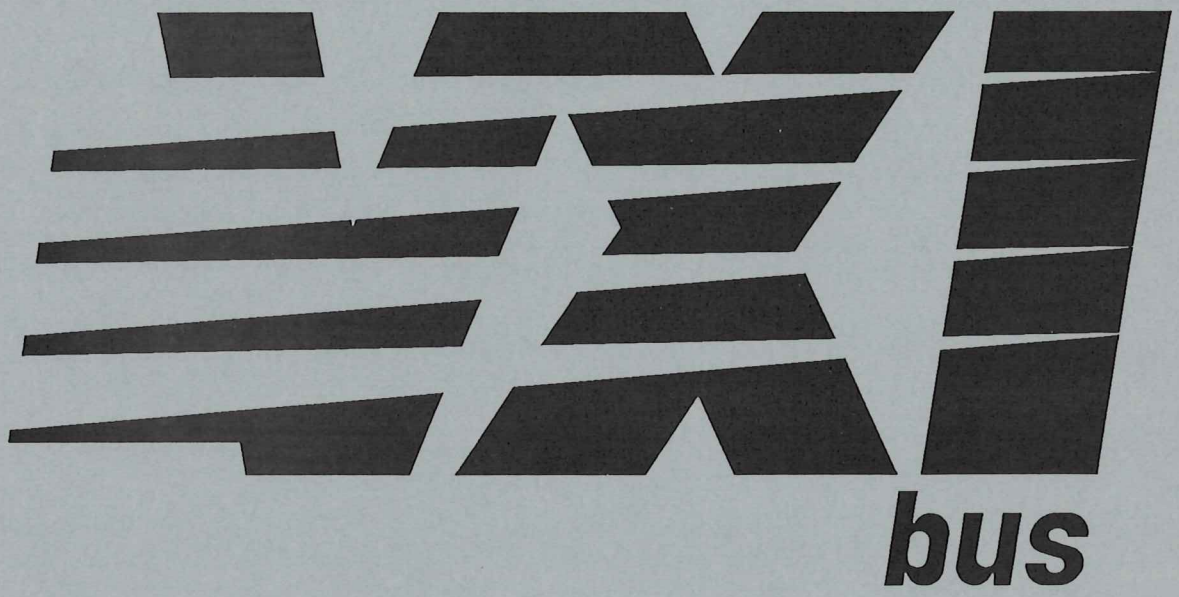
VXIbus term	Standard Mnemonic
Device Trigger	DTRIGger
Direct Current	DCurrent
Data Acknowledge	DTACK
Dynamic Configuration	DConfiguration
Dynamic Current	DCURrent
Dynamic Module Current	DMCurrent
Dynamic Module Input Current	DMICurrent
ECL Trigger	ECLTRG
ECL Trigger Lines n	ECLTRG*n*
Enable	ENABle
End Normal Operation	ENO
Extended Device	EDEVice
Extended Longword Serial Protocol	EWSProtocol
Extended START/STOP (Trigger Protocol)	ESTST
FAILED	FAILed
Fair Requester Protocol	FRPRotocol
Fast Handshake Transfer	FHS
Grant Device	GDEVice
Group Execute Trigger	GETrigger
Handler	HANDler
HARD RESET	HRESet
Hybrid Device	HDEVice
ID	ID
Identify Commander	ICOMmander
Induced Ripple/Noise	IRNoise
Inhibit	INHibit
INIT FAILED	IFAILed
INIT RESET	IRESet
INITIALIZE State	INITialize
Instrument Protocol	IPROTocol
INTERFACE DEVICE	IDEVice
Intermodule Separation Plane	ISPLane
Interrupt Acknowledge	IACK
Interrupt Control	ICONtrol
Interrupt Handler	IHANdler
Interrupt Handler n	IHANdler*n*
Interrupt Request line	IRQ*n*
Interrupt Status	ISTatus
Interrupter	INTerrupter
Interrupter n	INT*n*
LBUS00-11 lines	LBUS*nn*
LBUSA	LBUSA
LBUSAn	LBUSA*n*
LBUSC	LBUSC
LBUSCn	LBUSC*n*
Listener	LISTener
Load Ripple/Noise	LRNoise
Local Bus	LBUS
Local Bus Lockout Key	LBLKey
Local Lockout	LLOckout

VXIbus term	Standard Mnemonic
LOCATION MONITOR	LMONitor
Logical Address	LADDress
LONGWORD SERIAL Protocol	LWSProtocol
Mainframe Dynamic Current	MDCurrent
Mainframe Dynamic Output Current	MDOCurrent
Mainframe Peak Current	MPCurrent
MANAGER	MANager
Manufacturer Defined Subclasses	MDSubclasses
Manufacturer ID	MID
Master	MASTer
Memory Device	MDEVice
Message Based Device	MBDevice
Message Based	MBASed
Modid	MODID
Modid Line	MODID*n*
Module Current	MCURrent
MONITOR	MONitor
Multiple Acceptor Handshake Protocol	MAHandshake
NORMAL OPERATION Sub-State	NOPeration
Offset Address	OADDress
Offset	OFFSet
Output Enable	OENable
PASSED State	PASSed
Printed Circuit Board	PCBoard
Peak Current	PCURrent
Peak Module Current	PMCurrent
Periodic and Random Deviations	PARDeviations
Programmable Handler	PHANdler
Programmable Interrupter	PINTerrupter
Radiated Emissions	REMISsions
Radiated Susceptibility	RSUSceptibility
Read Handler Line	RHLine
Read Handlers	RHANdlers
Read Interrupter Line	RILine
Read Interrupters	RINTerrupter
Read MODID	RMODid
Read Only	RONLy
Read Protocol	RPRotocol
Read Protocol Error	RPERrror
Read Ready	RReady
Read Servant Area	RSARea
Read STB	RSTByte
REGISTER	REGister
Register Based	RBASed
Register Based Device	RBDevice
Release Device	RDEVice
Release On Acknowledge	ROAK
Remote	REMote
Remote With Lockout State	RWLS
Request False	REQF

VXIbus term	Standard Mnemonic
Request on No Request	RONR
Request Service	RQS
Request True	REQT
Reset	RESet
Resource Manager	RMANager
Response	RESPonse
Selective Device Clear	SDClear
SELF TEST	STESt
Semi-Synchronous (Trigger Protocol)	SEMI-SYNC
SERCLK	SERCLK
Serial Number High	SNHigh
Serial Number Low	SNLow
Servant	SERVant
Servant Area	SARea
Servant Area Size	SASize
Servant Device	SDEVice
Service Request	SRQ
Set Lock	SLOCk
Set Lower MODID	SLModid
Set Upper MODID	SUModid
Shared Memory Protocol	SMPRotocol
Signal	SIGNal
Single Acceptor Protocol	SAPRotocol
SLAVE	SLAVe
Slot n	SLOT*n*
SOFT RESET	SRESet
Source Handshake	SHANdshake
Serial Poll	SPOLL
START	STARt
Start/Stop (Trigger Protocol)	STST
STARX	STARX
STARX n=1-12	STARX*n*
STARY	STARY
STARY n=1-12	STARY*n*
Static Configuration	SConfiguration
Status	STATus
Status Byte	STB
STATUS/ID	STID
Standby	STDBY
Subclass	SUBClass
Sumbus	SUMBUS
Synchronous (Trigger Protocol)	SYNC
SYNC100	SYNC100
Sysfail	SYSFAIL
Sysfail Inhibit	SINHibit
Sysreset	SYSRESET
Talker	TALKer
Top Level Commander	TLCommander
Trigger	TRIGger
TTL Trigger	TTLTRG

VXIbus term	Standard Mnemonic
TTL Trigger Line	TTLTRG*n*
Unsupported Command	UCOMmand
Version Number	VNUMber
Word Serial Protocol	WSPRotocol
Write Ready	WReady

VMEbus Extensions for Instrumentation:



Common ASCII System Commands

Specification

VXI-5

REVISION: 1.0

June 23, 1991

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

This specification was developed in cooperation with the SCPI (Standard Commands for Programmable Instrumentation) Consortium. The ASCII commands defined by this document comply with the requirements of SCPI 1991.0. However SCPI defines many requirements beyond the scope of this specification. For more information on SCPI, see the SCPI specification.

VMEbus Extensions for Instrumentation: Common ASCII System Commands VXI-5 Revision 1.0 is authored by the VXIbus Consortium, Inc. and its sponsor members:

A/S Bruel & Kjaer
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments, Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce and distribute this document by any means for any purpose.

CONTENTS

A. INTRODUCTION 1
B. OVERVIEW 2
C. SELECT COMMAND 5
D. CONFIGURATION COMMANDS 6
E. REGISTER ACCESS COMMANDS 11
F. WORD SERIAL PROTOCOL SUPPORT COMMANDS 13

VMEbus Extensions for Instrumentation:

Common ASCII System Commands

A. INTRODUCTION

The VXIbus architecture defines several attributes which are common to all VXIbus systems. These are embodied in the system configuration and communication protocols. Among these protocols is an ASCII oriented communication protocol, the byte transfer protocol. This byte transfer protocol may be used between VXIbus message based devices, or routed to/from an IEEE 488 interface to external devices. It is often important for a user to access critical system configuration or access some of the low-level system protocols via one of the system components, such as an IEEE 488 to VXI interface device. This standard defines a set of ASCII commands which may be used to access these system capabilities in a common fashion. Use of these commands will enhance the interchangeability of VXIbus system components.

This specification is compatible with:

- VMEbus Extensions for Instrumentation System Specification, Revision 1.3, July 14, 1989, hereafter referred to as VXI-1 Rev 1.3.
- IEEE Std 488.2-1987 Codes, Formats, Protocols, and Common Commands for Use with ANSI/IEEE Std 488.1-1987 IEEE standard Digital Interface for Programmable Instrumentation. The IEEE, Inc.
- Standard Commands for Programmable Insrumentation (SCPI), Version 1991.0, April 1991. SCPI Consortium.

B. OVERVIEW

This specification describes ASCII commands which are issued to the system from an external host. The exact internal destination device and routing methods to that device are system specific. These commands are used for access to system configuration information, and for common capabilities. The terminology used in this specification is patterned after IEEE 488.2 and SCPI 1991.0..

The following commands are the standard set of VXIbus system commands which are ASCII encoded.

RULE B.0.1:
If a device implements one or more of these commands, the syntax of this section **SHALL** be used. The response syntax **SHALL** also follow this section.

The COMMON ASCII SYSTEM COMMANDS are organized into subsystems.

RULE B.0.2:
If a device implements a required command in a subsystem then it **SHALL** implement all the required commands in the subsystem.

RULE B.0.3:
If a device implements an optional command in a subsystem then it **SHALL** implement all the required commands in the subsystem.

RULE B.0.4:
If a device implements a command from any subsystem then it **SHALL** also implement the VXI:SEL command.

KEYWORD	PARAMETER FORM	NOTES	STATUS
VXI			
:CONFigure			new
:NUMber?		[query only]	new
:LADDress?		[query only]	new
:INFormation?		[query only]	new
:ALL?		[query only]	new
:VERBose?		[query only]	new
:ALL?		[query only]	new
:HIERarchy?		[query only]	new
:ALL?		[query only]	new
:VERBose?		[query only]	new
:ALL?		[query only]	new
:REGister			new
:READ?	<numeric_value>|<reg_name>,<numeric_value>	[query only]	new
:VERBose?	<numeric_value>|<reg_name>	[query only]	new
:WRITe	<numeric_value>|<reg_name>	[no query]	new
:RESet?		[query only]	new
:VERBose?		[query only]	new
:SELect	<numeric_value>		new
:WSProtocol			new
:COMMand			new
[:ANY]	<numeric_value>	[no query]	new
:AHILine	<numeric_value>,<numeric_value>	[no query]	new
:AILine	<numeric_value>,<numeric_value>	[no query]	new
:AMControl	<numeric_value>	[no query]	new
:ANO	<Boolean>	[no query]	new
:BAVailable	<Boolean>,<numeric_value>	[no query]	new
:BNO	<Boolean>	[no query]	new
:BRQ		[no query]	new
:CEVent	<Boolean>,<numeric_value>	[no query]	new
:CLR		[no query]	new
:CLOCk		[no query]	new
:CRESponse	<numeric_value>	[no query]	new
:ENO		[no query]	new
:GDEVice	<numeric_value>	[no query]	new
:ICOMmander	<numeric_value>	[no query]	new
:RDEVice	<numeric_value>	[no query]	new
:RHANdlers		[no query]	new
:RHLine	<numeric_value>	[no query]	new
:RILine	<numeric_value>	[no query]	new
:RINTerrupter		[no query]	new
:RMODid		[no query]	new
:RPERror		[no query]	new
:RPRotocol		[no query]	new
:RSTB		[no query]	new
:RSAREa		[no query]	new
:SLModid	<Boolean>,<numeric_value>	[no query]	new
:SLOCk		[no query]	new
:SUModid	<Boolean>,<numeric_value>	[no query]	new
:TRIGger		[no query]	new
:MESSage			new

KEYWORD	PARAMETER FORM	NOTES	STATUS
:RECeive?	<numeric_value>|<terminator>	[query only]	new
:SEND	<message_string>[,(END|NEND)]	[no query]	new
:QUERy			new
[:ANY?]	<numeric_value>	[query only]	new
:AHILine?	<numeric_value>,<numeric_value>	[query only]	new
:AILine?	<numeric_value>,<numeric_value>	[query only]	new
:AMControl?	<numeric_value>	[query only]	new
:ANO?		[query only]	new
:BNO?	<Boolean>	[query only]	new
:BRQ?		[query only]	new
:CEVent?	<Boolean>,<numeric_value>	[query only]	new
:CRESponse?	<numeric_value>	[query only]	new
:ENO?		[query only]	new
:RDEVice?	<numeric_value>	[query only]	new
:RHANdlers?		[query only]	new
:RHLine?	<numeric_value>	[query only]	new
:RILine?	<numeric_value>	[query only]	new
:RINTerrupter?		[query only]	new
:RMODid?		[query only]	new
:RPERror?		[query only]	new
:RPRotocol?		[query only]	new
:RSTB?		[query only]	new
:RSAREa?		[query only]	new
:SLModid?	<Boolean>,<numeric_value>	[query only]	new
:SUModid?	<Boolean>,<numeric_value>	[query only]	new
:RESPonse?		[query only]	new

C. SELECT COMMAND

VXI:SELect <logical_address>

This command specifies the logical address which is to be used by all subsequent commands in the VXI subsystem. Logical_address is specified as a numeric value in the range of 0 to 255. The query form returns the logical address as a <NR1> the value of -1 is reserved to indicate that no logical address has been selected. The *RST default value for logical_address is that no logical address is selected. All other commands which require a logical_address to be selected will respond with Error -221, "settings conflict", if no logical address is selected.

D. CONFIGURATION COMMANDS

VXI:CONFigure:NUMBer?

When issued to a resource manager, this command returns a <NR1> which is the number of devices in the system. If the command is received by a device which is not the resource manager, then the response to the command will contain the number of devices which are immediate servants to the destination device including the destination device. For example, a commander with 3 servants would return a value of 4, or a resource manager for a system of 4 devices would return a value of 5. The range of this integer numeric value is 1 to 256 inclusive. This is a required configuration command.

VXI:CONFigure:LADDress?

When issued to a resource manager, this command returns a comma separated list of <NR1> which are the logical addresses of the devices in the system. Each numeric value will be an integer in the range of 0 to 255 inclusive. The logical address of the device which is responding to the command will be the first item in the list. If the command is received by a device other than the resource manager, then the response to the command will contain the logical address of the destination device followed by a list of the devices which are immediate servants to the destination device. This is a required configuration command.

VXI:CONFigure:INFormation?

This command returns the static information about the selected logical address. The individual fields of the response are comma separated. If the information about the selected logical address is not available from the destination device, (That is, the requested device is not in the servant area of the destination device), then Error -224, "parameter error", will be set and no response data will be sent. This is a required configuration command. The command returns the following values:

Logical address: a <NR1>, between -1 and 255 inclusive. A -1 indicates that the device has no logical address.

manufacturer id: a <NR1>, between -1 and 4095 inclusive. -1 indicates that the device has no manufacturer id.

model code: a <NR1>, between -1 and 65535 inclusive. -1 indicates that the device has no model code

device class: a <NR1>, between 0 and 5 inclusive. Each value corresponds the the device's classification as follows:

0: VXIbus memory device.

1: VXIbus extended device.

2: VXIbus message based device.

3: VXIbus register based device.

4: Hybrid device.

5: Non-VXIbus device.

address space: a <NR1>, between 0 and 15 inclusive, which is the sum of the binary weighted codes of the address space(s) occupied by the device. The following values are defined:

1: The device has A16 registers.

2: The device has A24 registers.

4: The device has A32 registers.

8: The device has A64 registers.

A16 memory offset: the base address for any A16 registers (other than the VXIbus defined registers) which are present on the device. This is a <NR1> in the range of -1 to 65535. -1 is used to indicate that the device has no A16 memory.

A24 memory offset: the base address for any A24 registers which are present on the device. This is a <NR1> in the range of -1 to 16777215. -1 is used to indicate that the device has no A24 memory.

A32 memory offset: the base address for any A32 registers which are present on the device. This is a <NR1> in the range of -1 to 4294967295. -1 is used to indicate that the device has no A32 memory.

A16 memory size: the number of bytes reserved for any A16 registers (other than the VXIbus defined registers) which are present on the device. This is a <NR1> in the range of -1 to 65535. -1 is used to indicate that the device has no A16 memory.

A24 memory size: the number of bytes reserved for any A24 registers which are present on the device. This is a <NR1> in the range of -1 to 16777215. -1 is used to indicate that the device has no A24 memory.

A32 memory size: the number of bytes reserved for any A32 registers which are present on the device. This is a <NR1> in the range of -1 to 4294967295. -1 is used to indicate that the device has no A32 memory.

Slot number: a <NR1>, between -1 and the number of slots which exist in the cage. -1 indicates that the slot which contains this device is unknown.

Slot 0's logical address: a <NR1>, between -1 and 255 inclusive. -1 indicates that Slot 0 device associated with this device is unknown.

subclass: a <NR1> representing the contents of the subclass register. -1 indicates that the subclass register is not defined for this device.

attribute: a <NR1> representing the contents of the attribute register. -1 indicates that the attribute register is not defined for this device.

manufacturer specific comment: This up to 80 character quoted string contains manufacturer specific data. It is sent with a 488.2 string response data format.

VXI:CONFigure:INFormation:ALL?

When issued to a resource manager, this command returns the static information about all logical addresses. If the command is received by a device which is not the resource manager, it returns the static information about the destination device followed by all of its immediate servants. The information is returned in the order specified in the response to VXI:CONF:LADD?. (the information about multiple logical addresses will be semi-colon separated and follow the IEEE 488.2 response message format). The individual fields of the output are comma separated. This is an optional configuration command.

VXI:CONFigure:INFormation:VERBose?

This command has the same functionality as VXI:CONFigure:INFormation?. However it returns a quoted string indicating the device information for the selected device. The format of this string is manufacturer specific. This command is intended to allow for a human readable form of the response to VXI:CONF:INF?. This is an optional configuration command.

VXI:CONFigure:INFormation:VERBose:ALL?

This command has the same functionality as VXI:CONFigure:INFormation:ALL?. However it returns a semi-colon separated sequence of quoted strings indicating the device information for all devices. The format of each string is manufacturer specific. This command is intended to allow for a human readable form of the response to VXI:CONF:INF:ALL?. The format of this string is manufacturer specific. This is an optional configuration command.

VXI:CONFigure:HIERarchy?

This command returns current hierarchy configuration information about the selected logical address. The individual fields of the response are comma separated. If the information about the selected logical address is not available from the destination device, (That is, the requested device is not in the servant area of the destination device), then Error -224, "parameter error", will be set and no response data will be sent. This is a required configuration command. The output fields are defined as follows:

Logical address: a <NR1>, between -1 and 255 inclusive. A -1 indicates that the device has no logical address.

commander's logical address: a <NR1>, between -1 and 255 inclusive. -1 indicates that this device has no commander or the commander is unknown.

Interrupt Handlers: a comma separated list of 7 <NR1>, between 0 and 7 inclusive. Interrupt lines 1-7 are mapped to the individual return values. 0 is used to indicate that the particular interrupt handler is not configured. A set of return values of 0,0,0,5,2,0,6 would indicate that the device is configured as follows

handler 4 is configured to handle interrupts on line 5
handler 5 is configured to handle interrupts on line 2
handler 7 is configured to handle interrupts on line 6
handlers 1,2,3,6 are not configured

Interrupters: a comma separated list of 7 <NR1>, between 0 and 7 inclusive. Interrupt lines 1-7 are mapped to the individual return values. 0 is used to indicate that the particular interrupter is not configured. A set of return values of 0,0,0,5,2,0,6 would indicate that the device is configured as follows

interrupter 4 is configured to interrupt on line 5
interrupter 5 is configured to interrupt on line 2
interrupter 7 is configured to interrupt on line 6
interrupters 1,2,3,6 are not configured

Pass/Failed: a <NR1> which contains the pass/fail status of the specified device encoded as follows: FAIL=0, IFAIL=1, PASS=2, READY=3.

manufacturer specific comment: This up to 80 character quoted string contains manufacturer specific data. It is sent with a 488.2 string response data format.

VXI:CONFigure:HIERarchy:ALL?

When issued to a resource manager, configuration information about about all logical addresses. If the command is received by a device which is not the resource manager, it returns the current hierarchy configuration information about the destination device followed by all of its immediate servants. The information is returned in the order specified in VXI:CONF:LADD?. (the information about multiple logical addresses will be semi-colon separated and follow the IEEE 488.2 response message format). The individual fields of the output are comma separated. This is an optional configuration command.

VXI:CONFigure:HIERarchy:VERBose?

This command has the same functionality as VXI:CONFigure:HIERarchy?. However it returns a quoted string indicating the device information for the selected This command is intended to allow for a human readable form of the response to VXI:CONF:HIER?. The format of this string is manufacturer specific. This is an optional configuration command.

VXI:CONFigure:HIERarchy:VERBose:ALL?

This command has the same functionality as VXI:CONFigure:HIERarchy:ALL?. However it returns a semi-colon separated sequence of strings indicating the device information for all devices. The format of this string is manufacturer specific. This command is intended to allow for a human readable form of the response to VXI:CONF:HIER:ALL?. This is an optional configuration command.

E. REGISTER ACCESS COMMANDS

VXI:REGister:READ? <register>

This command returns the contents of the specified 16 bit register at the selected logical address as a <NR1>. This is a required register access command. The register is specified as the byte address of the desired register or as optionally the register name. It has values of all even numbers from 0 to 62 inclusive (as a <numeric_value>) or the following (optional) words:

A24Low: A24 Pointer Low register (18)

A24High: A24 Pointer High register (16)

A32Low: A32 Pointer Low register (22)

A32High: A32 Pointer High register (20)

ATTRibute: Attribute register (8)

DHIGh: Data High register (12)

DLOW: Data Low register (14)

DTYPe: Device Type register (2)

ICONtrol: Interrupt Control register (28)

ID: ID register (0)

ISTatus: Interrupt Status register (26)

MODid: MODID register (8)

OFFSet: Offset register (6)

PROTotocol: PROTocol register (8)

RESPonse: RESPonse register (10)

SNLow: Serial Number Low register (12)

SNHigh: Serial Number High register (10)

STATus: Status register (4)

SUBClass: Subclass register (30)

VNUMber: Version Number register (14)

VXI:REGister:READ:VERBose? <register>

This command has the same functionality as VXI:REGister:READ?. However it returns a quoted string indicating the register contents for the selected device. The format of this string is manufacturer specific. This command is intended to allow for a human readable form of the response to VXI:REG:READ? which could include additional decoding of bits in specific registers. This is an optional register access command.

VXI:REGister:WRITe <register>, <data> This command writes data to the specified register on the selected logical address. This is a required register access command. The Data is a 16 bit value specified as a <numeric_value> in the range of -32768 to 32767. The register is specified as the byte address of the desired register or as optionally the register name. It has values of all even numbers from 0 to 62 inclusive (expressed as a <numeric_value>) or the following (optional) words:

CONTrol: Control register (4)

DEXTended: Data Extended register (10)

DHIGh: Data High register (12)

DLOW: Data Low register (14)

ICONtrol: Interrupt Control register (28)

MODid: MODID register (8)

LADDress: Logical Address register (0)

OFFSet: Offset register (6)

SIGNal: Signal register (8)

VXI:RESet?

This command resets the selected logical address. SYSFAIL generation is inhibited while the device is in the self test state. The command waits for five seconds or until the selected device has indicated passed (whichever occurs first). If the device passes its self test, then SYSFAIL generation will be re-enabled. If the device fails its self test, then SYSFAIL generation will remain inhibited. The return value from this command is the state of the selected device after it has been reset. The command returns an <NR1> which is encoded as follows: FAIL=0, PASS=2, READY=3. The state of the A24/A32 enable bit is not altered by this command. This is a required register access command.

VXI:RESet:VERBose?

This command has the same functionality as VXI:RESet?. However it returns a quoted string indicating the state of the specified device. The format of this string is manufacturer specific. This is an optional register access command.

F. WORD SERIAL PROTOCOL SUPPORT COMMANDS

VXI:WSPRotocol:COMMand[:ANY] <data>

This command sends the specified word serial command to the selected logical address. The data field, a numeric_value (which may be a non-decimal numeric value), specifies the command to be sent. The response of a word serial query which is sent with this command can be read with the VXI:WSPRotocol:RESPonse? command. This is a required Word Serial Support command.

VXI:WSPRotocol:COMMand:AHLine <hand_id>,<line_number>

This command sends an Assign Handler Line command to the selected logical address. The hand_id field has a value in the range of 1 to 7 inclusive, and line_number field has a value in the range of 0 to 7 inclusive. They are both specified with numeric_values (which may be non-decimal numeric values). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:AILine <int_id>,<line_number>

This command sends an Assign Interrupter Line command to the selected logical address. The int_id field has a value in the range of 1 to 7 inclusive, and line_number field has a value in the range of 0 to 7 inclusive. They are both specified with numeric_values (which may be non-decimal numeric values). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:AMControl <response_mask>

This command sends an Asynchronous Mode Control command to the selected logical address. The response mask field has a value in the range of 0 to 15 inclusive, and is defined in VXI rev. 1.3. It is specified with a numeric_value (which may be a non-decimal numeric value). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:ANO

This command sends an Abort Normal Operation command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:BAVailable <Boolean>,<byte>

This command sends a Byte Available command to the selected logical address. The Boolean field selects whether the END bit is set in the command. The byte field has a value in the range of 0 to 255 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:BNO <Boolean>

This command sends an Begin Normal Operation command to the selected logical address. The Boolean field selects whether the Top_level bit is set in the command. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:BRQ

This command sends an Byte Request command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:CEVent <Boolean>,<event_number>

This command sends a Control Event command to the selected logical address. The Boolean field selects whether the Enable bit is set in the command. The event_number field has a value in the range of 0 to 127 inclusive. It is specified with a numeric value (which may be a non-decimal numeric value). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:CLR

This command sends an Clear command to the selected logical address. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:CLOCk

This command sends an Clear Lock command to the selected logical address. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:CRESponse <response_mask>

This command sends a Control Response command to the selected logical address. The response_mask field has a value in the range of 0 to 127 inclusive, and is defined in VXI rev. 1.3. It is specified with a numeric value (which may be a non-decimal numeric value). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:ENO

This command sends an End Normal Operation command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:GDEVice <logical_address>

This command sends a Grant Device command to the selected logical address. The logical_address field has a value in the range of 0 to 255 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:ICOMmander <logical_address>

This command sends a Identify Commander command to the selected logical address. The logical_address field has a value in the range of 0 to 255 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RDEVice <logical_address>

This command sends a Release Device command to the selected logical address. The logical_address field has a value in the range of 0 to 255 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RHANdlers

This command sends a Read Handlers command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RHLine <hand_id>

This command sends a Read Handler Line command to the selected logical address. The hand_id field has a value in the range of 1 to 7 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RILine <int_id>

This command sends a Read Interrupter Line command to the selected logical address. The int_id field has a value in the range of 1 to 7 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RINTerrupter

This command sends a Read Interrupters command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RMODid

This command sends a Read MODID command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RPERror

This command sends a Read Protocol Error command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RPRotocol

This command sends a Read Protocol command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RSTB

This command sends a Read Status Byte command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:RSARea

This command sends a Read Servant Area command to the selected logical address. The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:SLModid <Boolean>,<MODID 6-0>

This command sends a Set Lower MODID command to the selected logical address. The Boolean field selects whether the Enable bit is set in the command. The MODID 6-0 field has a value in the range of 0 to 127 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:SLOCk

This command sends a Set Lock command to the selected logical address. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:SUModid <Boolean>,<MODID 12-7>

This command sends a Set Upper MODID command to the selected logical address. The Boolean field selects whether the Enable bit is set in the command. The MODID 12-7 field has a value in the range of 0 to 63 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value).

The response to this command can be read with the VXI:WSPRotocol:RESPonse? command. This is an optional Word Serial Support command.

VXI:WSPRotocol:COMMand:TRIGger

This command sends a Trigger command to the selected logical address. This is an optional Word Serial Support command.

VXI:WSPRotocol:MESSage:SEND <message_string> [,(END|NEND)]

This command sends the specified message string to the selected logical address. The string is sent using the word serial protocol with the byte transfer protocol. The last byte of the string is sent with the End bit set unless NEND is specified. This is a required Word Serial Support command.

VXI:WSPRotocol:MESSage:RECeive? <count>|<terminator>

This command receives a message from the selected logical address using both the word serial protocol and the byte transfer protocol. The command will always terminate on the End bit being set. Additional termination options are on a specified number of bytes (count), or on a match to a particular terminator (That is, LF, CRLF, END). The response is returned as a string. This is a required Word Serial Support command.

VXI:WSPRotocol:QUERy[:ANY]? <data>

This command sends the specified word serial query to the selected logical address. The data field is a numeric_value (which may be a non-decimal numeric value) which specifies the query to be sent. The returned value is the response to the word serial query and is a <NR1>. This command obeys the byte transfer protocol. This is a required Word Serial Support command.

VXI:WSPRotocol:QUERy:AHLine? <hand_id>,<line_number>

This command sends an Assign Handler Line command to the selected logical address. The hand_id field has a value in the range of 1 to 7 inclusive, and line_number field has a value in the range of 0 to 7 inclusive. They are both specified with numeric_values (which may be non-decimal numeric values). The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:AILine? <int_id>,<line_number>

This command sends an Assign Interrupter Line command to the selected logical address. The int_id field has a value in the range of 1 to 7 inclusive, and line_number field has a value in the range of 0 to 7 inclusive. They are both specified with numeric_values (which may be non-decimal numeric values). The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:AMControl? <response_mask>

This command sends an Asynchronous Mode Control command to the selected logical address. The response_mask field has a value in the range of 0 to 15 inclusive, and is defined in VXI rev. 1.3. It is specified with a numeric_value (which may be a non-decimal numeric value). The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:ANO?

This command sends an Abort Normal Operation command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:BNO? <Boolean>

This command sends an Begin Normal Operation command to the selected logical address. The Boolean field selects whether the Top_level bit is set in the command. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:BRQ?

This command sends an Byte Request command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:CEVent? <Boolean>,<event_number>

This command sends a Control Event command to the selected logical address. The Boolean field selects whether the Enable bit is set in the command. The event_number field has a value in the range of 0 to 127 inclusive. It is specified with numeric_value (which may be a non-decimal numeric value). The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:CRESponse? <response_mask>

This command sends a Control Response command to the selected logical address. The response_mask field has a value in the range of 0 to 127 inclusive, and is defined in VXI rev. 1.3. It is specified with a numeric_value (which may be a non-decimal numeric value). The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:ENO?

This command sends an End Normal Operation command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RDEVice? <logical_address>

This command sends a Release Device command to the selected logical address. The logical_address field has a value in the range of 0 to 255 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value). The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RHANdlers?

This command sends a Read Handlers command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RHLine? <hand_id>

This command sends a Read Handler Line command to the selected logical address. The hand_id field has a value in the range of 1 to 7 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value).

The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RILine? <int_id>

This command sends a Read Interrupter Line command to the selected logical address. The int_id field has a value in the range of 1 to 7 inclusive. It is specified with a numeric value (which may be a non-decimal numeric value).

The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RINTerrupter?

This command sends a Read Interrupters command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RMODid?

This command sends a Read MODID command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RPERror?

This command sends a Read Protocol Error command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RPRotocol?

This command sends a Read Protocol command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RSTB?

This command sends a Read Status Byte command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:RSARea?

This command sends a Read Servant Area command to the selected logical address. The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:SLModid? <Boolean>,<MODID 6-0>

This command sends a Set Lower MODID command to the selected logical address. The Boolean field selects whether the Enable bit is set in the command. The MODID 6-0 field has a value in the range of 0 to 127 inclusive. It is specified with a numeric_value or (which may be a non-decimal numeric value).

The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:QUERy:SUModid? <Boolean>,<MODID 12-7>

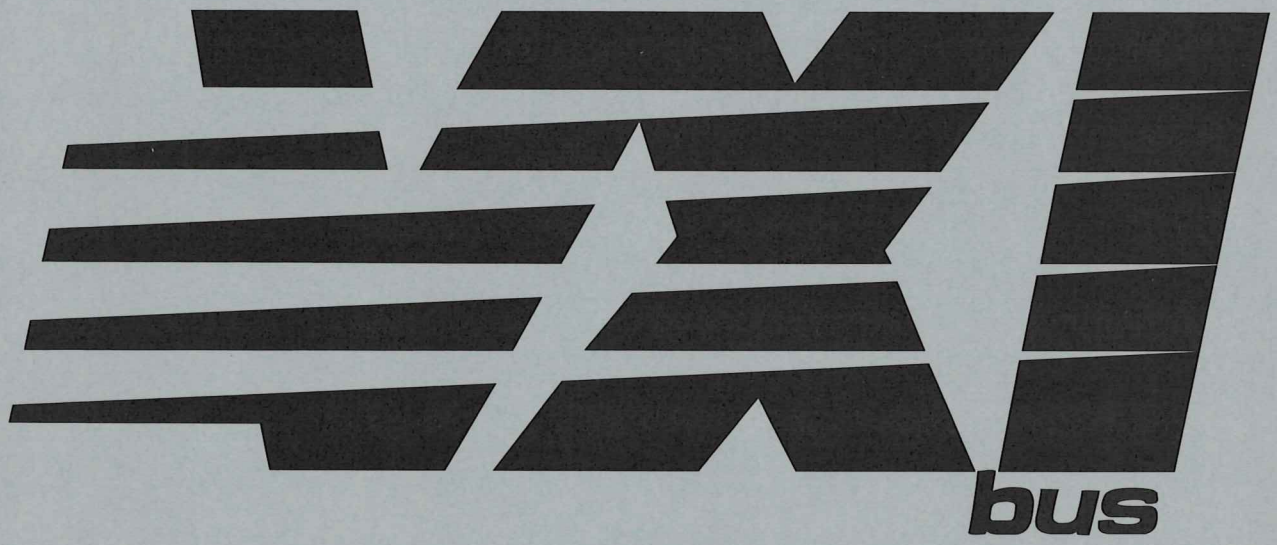
This command sends a Set Upper MODID command to the selected logical address. The Boolean field selects whether the Enable bit is set in the command. The MODID 12-7 field has a value in the range of 0 to 63 inclusive. It is specified with a numeric_value (which may be a non-decimal numeric value).

The returned value is the response to the command and is a <NR1>. This is an optional Word Serial Support command.

VXI:WSPRotocol:RESPonse?

This command returns one word of data from the data low register on the selected logical address. This command obeys the Byte transfer protocol. The data is returned as a <NR1>. This is a required Word Serial Support command.

VMEbus Extensions for Instrumentation



Mainframe Extender Specification

VXI-6

Revision 1.0

February 7, 1991

Tektronix
Test and Measurement

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extensions for Instrumentation, Mainframe Extender Specification VXI-6, Revision 1.0, is authored by the VXIbus Consortium, Inc. and its sponsor members:

A/S Bruel & Kjaer
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce this document by any means for any purpose.

CONTENTS

A. INTRODUCTION 1

B. OVERVIEW 2

B.1 SCOPE 2

B.2 TERMINOLOGY 3

C. MULTIFRAME SYSTEM INTERCONNECTION 4

C.1 TOPOLOGY 4

C.2 MAINFRAME-to-INTERCONNECT BUS INTERFACE 5

C.3 STAND-ALONE DEVICES 5

C.4 ADDRESSING AND DATA TRANSFERS between MAINFRAMES 5

C.5 BUS TIMER OPERATION 7

C.6 BUS ARBITRATION OPERATION 7

C.6.1 Interlocked Arbitration 7

C.6.2 Non-Interlocked Arbitration 7

C.7 INTERRUPTS and INTERRUPT HANDLING between MAINFRAMES 8

C.8 TTL TRIGGERING between MAINFRAMES 9

C.9 ECL TRIGGERING between MAINFRAMES 10

C.10 UTILITY BUS OPERATION between MAINFRAMES 11

D. MAINFRAME EXTENDER 12

D.1 MAINFRAME EXTENDER REGISTERS 12

D.1.1 Configuration Registers 13

D.1.2 MODID Register 13

D.1.3 Logical Address Window Register 14

D.1.4 A16 Window Register 15

D.1.5 A24 Window Register 16

D.1.6 A32 Window Register 17

D.1.7 Interrupt Configuration Register 18

D.1.8 TTL Trigger Configuration Register 19

D.1.9 ECL Trigger Configuration Register 20

D.1.10 Utility Configuration Register 21

D.1.11 Reserved Registers 22

D.1.12 Subclass Register 22

D.1.13 Device Dependent Registers 22

D.2 INITIALIZATION AND CONTROL 22
E. SYSTEM CONFIGURATION 24
E.1 LOGICAL ADDRESS WINDOW CONFIGURATION 24
E.1.1 System Configuration Example 25
E.2 A16 WINDOW CONFIGURATION 29
E.3 A24 WINDOW CONFIGURATION 30
E.4 A32 WINDOW CONFIGURATION 30
E.5 INTERRUPT CONFIGURATION 30
E.6 TRIGGER CONFIGURATION 31
E.7 UTILITY CONFIGURATION 31
F. MULTIFRAME RESOURCE MANAGER OPERATION 32
F.1 LOGICAL ADDRESS WINDOW CONFIGURATION and DEVICE IDENTIFICATION 32
F.1.1 An Example 33
F.2 A16, A24, and A32 SPACE ADDRESS MAP CONFIGURATION 34
F.3 SYSTEM SELF-TEST MANAGEMENT 35
F.4 SYSTEM HIERARCHY INITIALIZATION 35
F.5 INTERRUPT RESOURCE CONFIGURATION 35
F.6 TTL and ECL TRIGGER and UTILITY RESOURCE CONFIGURATION 35
F.7 INITIATION of NORMAL OPERATION 36
APPENDIX I. MULTIFRAME RESOURCE MANAGER OPERATION 37

LIST OF FIGURES

Figure C.1. Interconnection Topology Example 4
Figure C.2. Data Transfers between Mainframes 6
Figure C.3. Data Transfer Bus Arbitration Deadlock 8
Figure D.1. Mainframe Extender Register Map 12
Figure E.1. Example System Topology 25
Figure E.2. Example Logical Address Map 29
Figure I.1. Register Overview for Mainframe Extender 38

LIST OF TABLES

Table E.1. Logical Address Span Example 26
Table E.2. Logical Address Window Configuration Example 27
Table E.3. Logical Address Range Example 28

VMEbus Extensions for Instrumentation
Mainframe Extender Specification

A. INTRODUCTION:

This document is an addendum to the VXIbus System Specification that defines the standard VXIbus Extender device. It also defines the architecture of multiframe VXIbus systems.

This specification is compatible with:

- VMEbus Extensions for Instrumentation System Specification, Revision 1.3, July 14, 1989, hereafter referred to as VXI-1, Rev. 1.3.

B. OVERVIEW:

The purpose of this document is to specify a standard interface for extending the VXIbus System Specification (VXI-1, Rev. 1.3) to provide for transparent communication and synchronization between devices in multiple-mainframe systems. The VMEbus and VXIbus extensions defined by this specification include:

- Addressing and Data Transfers
  - VXIbus Logical Address space
  - VMEbus A16/A24/A32 address spaces
  - VMEbus D08(EO)/D16/D32 data paths
- Interrupts and Interrupt Handling
- Inter-Device Synchronization
  - VXIbus TTL triggers
  - VXIbus ECL triggers
- Utility Bus
  - ACFAIL*
  - SYSFAIL*
  - SYSRESET*
- Multiframe System Configuration
- Multiframe Resource Management

The approach to inter-device communication is to partition each address space among the mainframes in a structured fashion, such that the system memory map is identical from the point of view of each device in the system. All of the VMEbus spaces are extended. Although the VXIbus Logical Address space is located in the upper 16 kilobytes of VMEbus A16 space, it has a separate extension mechanism. The lower 48 kilobytes of VMEbus A16 space is typically used by non-VXI devices (VMEbus devices, for example).

Simple routing mechanisms for interrupts, triggers, and the utility bus signals ACFAIL*, SYSFAIL*, and SYSRESET* are defined.

A multiframe VXIbus system has multiple Slot 0 devices, but only one global Resource Manager. In addition to the extension mechanisms, this document addresses some of the issues and algorithms associated with multiframe system configuration and resource management.

B.1 SCOPE

This specification defines several aspects of multiframe VXIbus systems, including:

- Interconnection topology
- Integration of stand-alone instruments and peripherals
- Addressing and data transfers
- Interlocked and non-interlocked arbitration
- Interrupts and Interrupt handling
- TTL and ECL triggering
- Utility bus operation
- Mainframe extender register set and behavior

- System configuration
- Multiframe Resource Manager operation

This revision of the specification does not define a standard interface for extending the VMEbus signals SERCLK, SERDAT, and SYSCLK, or the VXIbus signals SUMBUS, STARX/Y, CLK100, and SYNC100. These may be added in the future when and if it becomes appropriate to do so.

VXI-1, Rev. 1.3 recommends that Slot 0 devices allow CLK10 to be derived from an external frequency source, effectively providing for extension of the CLK10 signal.

This specification is based on a model in which the various mainframes are connected by some physical media (referred to as *interconnect buses*). These interconnect buses have similar functionality to the VXIbus. The specification does not, however, define the implementation details of the interconnect buses.

Interconnect buses may be point-to-point or multidrop. They support the VXIbus address and data path extensions, and may optionally support the interrupt, trigger, and utility bus (ACFAIL*, SYSFAIL*, and SYSRESET*) extensions.

B.2 TERMINOLOGY

The following terms are used in this specification, and are defined as follows:

bus	A VXIbus mainframe or interconnect bus
root bus	The bus to which the device at Logical Address 0 is connected
local	On the bus under consideration
remote	On a bus other than the one under consideration
outward	In the direction leading out of a VXIbus mainframe
inward	In the direction leading into a VXIbus mainframe
upward	In the direction toward the root bus
downward	In the direction away from the root bus
node	VXIbus mainframe or stand-alone device
root node	The physical location of the Resource Manager (Logical Address 0)
parent node	The directly connected node in the direction toward the root bus
child node	A directly connected node in the direction away from the root bus
parent bus	The directly connected bus in the direction toward the root bus
child bus	A directly connected bus in the direction away from the root bus
parent bus extender	The mainframe extender that connects the local bus to the parent bus
child bus extender	A mainframe extender that connects the local bus to child bus
descendant node	Any node in the subtree of the node under consideration
descendant bus	Any bus in the subtree of the bus under consideration
ancestor bus	Any bus between the root bus and the bus under consideration

C. MULTIFRAME SYSTEM INTERCONNECTION:

C.1 TOPOLOGY

The multiframe system interconnection topology is an arbitrary tree. This is a very flexible approach, because many topologies are equivalent to the arbitrary tree topology, that is, they can be redrawn in the arbitrary tree form. The primary characteristic of a tree topology is that there are no circular connection paths in its interconnection net. Two examples of illegal topologies are:

1. A connection net in which any subset of the nodes are connected in a ring, and
2. Two or more connections between a pair of nodes.

The nodes in a multiframe VXIbus system are either VXIbus mainframes or stand-alone devices.

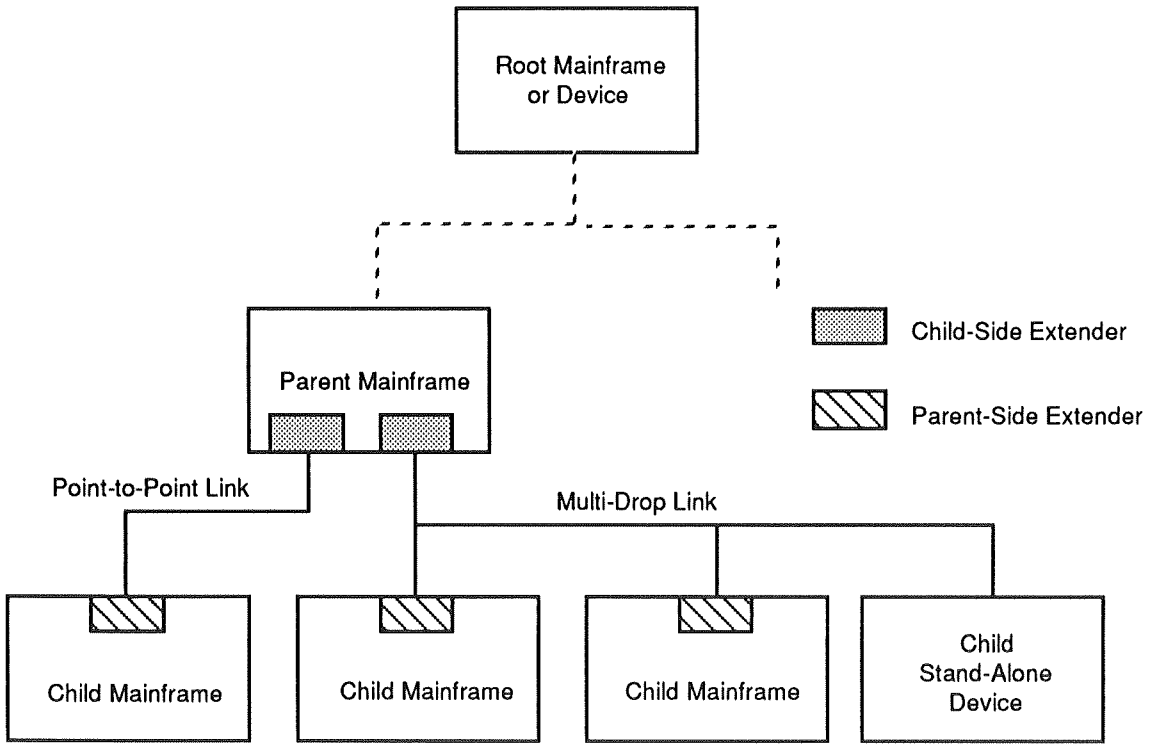


Figure C.1. Interconnection Topology Example.

In an arbitrary tree topology, any node can be defined to be the root. In the context of multiframe VXIbus systems, however, the root node is the physical location of the Resource Manager (Logical Address 0).

When considering the relationship between any pair of nodes, the node closer to the root is called the *parent* node. The node farther from the root is called the *child* node. The *descendants* of a node consist of all nodes connected to the tree through that node.

Nodes are connected by interconnect buses. Interconnect buses may be point-to-point (one parent, one child) or multidrop (one parent, multiple children). The interconnect buses support the address and data path extensions, and may optionally support interrupt, trigger, and utility bus (ACFAIL*, SYSFAIL*, and SYSRESET*) extensions.

C.2 MAINFRAME-to-INTERCONNECT BUS INTERFACE

A device called a *Mainframe Extender* interfaces a VXIbus mainframe to an interconnect bus. A Mainframe Extender routes bus transactions from the VXIbus to the interconnect bus (in the outward direction), or from the interconnect bus to the VXIbus (in the inward direction). A Mainframe Extender has a set of registers that defines the routing mechanisms for data transfers, interrupts, triggers and utility bus signals. These registers are accessible from either the VXIbus or from the interconnect bus. The register set is described in Section D, "Mainframe Extender."

Mainframe Extenders may have VXIbus Slot 0 capability.

An extender that interfaces a mainframe to the interconnect bus connected to its parent node is called a *parent-side* extender. An extender that interfaces a mainframe to an interconnect bus connected to any of its child nodes is called a *child-side* extender. Each mainframe (except the root) has one parent-side extender and any number of child-side extenders.

C.3 STAND-ALONE DEVICES

Some instruments and peripherals will not fit into the VXIbus form factor. This specification accommodates these devices by providing for them to connect directly to an interconnect bus. *Stand-alone* devices can have the same register definition and programming interface as VXIbus devices.

Stand-alone devices, however, cannot be dynamically configured because the MODID signals are only defined within mainframes.

C.4 ADDRESSING AND DATA TRANSFERS between MAINFRAMES

The programming model for VXIbus systems is a set of four memory spaces defined by the VMEbus and VXIbus specifications (VXIbus Logical Address, and VMEbus A16, A24, and A32).

Each Mainframe Extender has a window for each address space through which data transfers between local and remote devices take place. The tree structure allows each extender's windows to be configured in such a manner that the system memory map appears the same from every geographical location in the system.

The Resource Manager can auto-configure the Logical Address, A24, and A32 windows in a manner compatible with the memory requirements of the VXIbus devices in the system. Alternatively, the windows can be statically configured according to a nonvolatile table provided to the Resource Manager.

Static configuration accommodates non-VXI devices that have no means to indicate the type or size of the space they require. A16 space must be statically configured, because no mechanism for auto-configuring A16 space is defined in VXI-1, Rev. 1.3.

As with a single mainframe system, data transfers between mainframes are accomplished with read and write cycles. Data bus cycles that address a slave device in a remote mainframe are routed through a window, across an interconnect bus, and through the window of the Mainframe Extender located in the remote mainframe. The remote Mainframe Extender generates a remote VMEbus cycle that performs the slave access. The slave generates the handshake signal (DTACK* or BERR*) that the interconnect bus propagates back to the bus master.

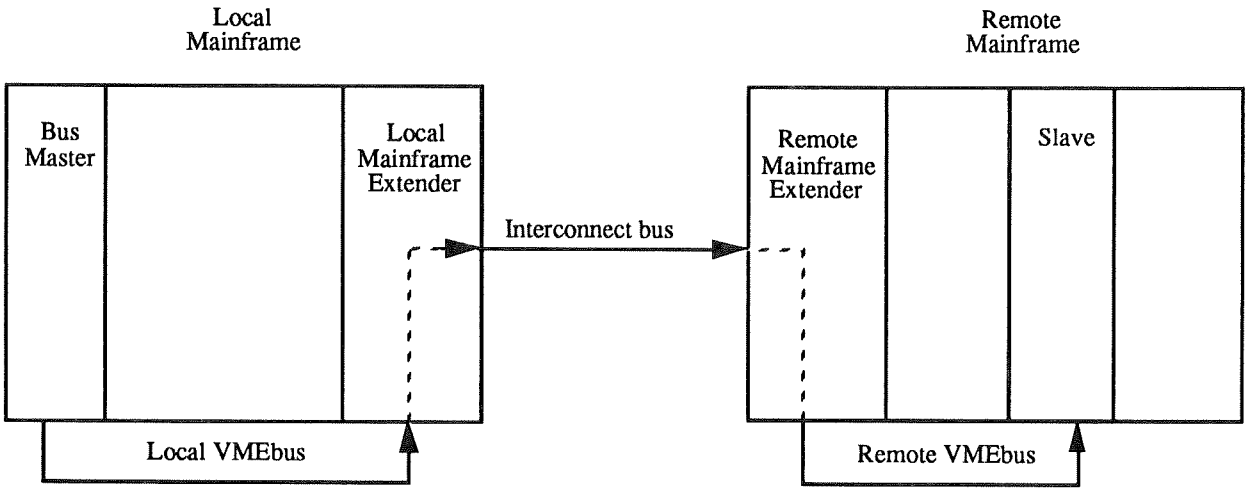


Figure C.2. Data Transfers between Mainframes.

RULE C.1:

Mainframe Extenders and interconnect buses **SHALL** transparently route bus cycles of the following types:

- Addressing modes
  - A16 Supervisory
  - A16 Non-Privileged
  - A24 Supervisory Program
  - A24 Supervisory Data
  - A24 Non-Privileged Program
  - A24 Non-Privileged Data
- Data transfer modes
  - D16
  - D08E
  - D08O

RECOMMENDATION C.1:

Mainframe Extenders and interconnect buses should route A32 bus cycles.

RULE C.2:

IF a Mainframe Extender or interconnect bus routes any A32 bus cycles,
THEN it **MUST** transparently route Supervisory Program, Supervisory Data, Non-Privileged Program, and Non-Privileged Data A32 bus cycles.

RECOMMENDATION C.2:

Mainframe Extenders and interconnect buses should transparently route D32 bus cycles.

PERMISSION C.1:

Mainframe Extenders and interconnect buses **MAY** route A24 and A32 Block Transfer cycles.

PERMISSION C.2:

Mainframe Extenders and interconnect buses **MAY** support addressing and data transfer modes defined in the future, such as the A64 and D64 modes defined in VMEbus Revision D.

RECOMMENDATION C.3:
Mainframe Extenders and interconnect buses should support both inward and outward access. Bus masters in both the local and remote mainframes should be able to access slaves in all other mainframes.

RULE C.3:
Mainframe Extenders and interconnect buses **SHALL** transparently preserve the addressing and data transfer modes between mainframes. For example, a D16 access generated by a bus master may not be broken up into two D08 transfers on the remote bus.

C.5 BUS TIMER OPERATION

The segmentation of the buses in multiframe systems limits the extent of any data transfer cycle to only those buses directly involved in the data transfer. In other words, none of the address, data, or control signals will be propagated onto buses not involved in the transfer. In many cases, the activity will be confined to a single bus. A Bus Timer on a non-participating bus will not be able to terminate an unacknowledged cycle.

RECOMMENDATION C.4:
Every bus should have a local Bus Timer.

C.6 BUS ARBITRATION OPERATION

Bus arbitration in a multiframe system may be interlocked or non-interlocked.

C.6.1 Interlocked Arbitration

With interlocked arbitration, the Mainframe Extenders and interconnect buses use a global bus arbitration scheme in which all data transfer bus (DTB) resources are owned by one bus master at a time. This approach prevents bus resource deadlock conditions. The advantage of this approach is transparent inter-operability between the mainframe extension mechanism and all VMEbus masters. The disadvantages of interlocked systems are:

- The number of independent bus resources does not grow with the system size.
- Global bus arbitration adds considerable overhead to each VMEbus access. The bus arbitration overhead increases with system size.

RULE C.4:
Mainframe Extenders and interconnect buses **SHALL** support interlocked bus arbitration.

C.6.2 Non-Interlocked Arbitration

With non-interlocked arbitration, the bus arbitration within each mainframe is independent of the arbitration in other mainframes, except when a bus master accesses a slave in a remote mainframe. This approach provides independent bus resources that can be shared by each group of associated devices, and removes the restriction on overall system bus bandwidth associated with the interlocked approach.

When a bus master in a non-interlocked system accesses a slave in another mainframe, it must first arbitrate for the local VMEbus and initiate a bus cycle (as it would normally do). The local Mainframe Extender recognizes that the address falls within one of its address windows and arbitrates for mastership of the interconnect bus and initiates an interconnect bus cycle. In response to the interconnect bus cycle, the remote Mainframe Extender generates a request for the remote VMEbus.

A deadlock condition can occur if the remote VMEbus has already been granted to a second bus master that has initiated a bus cycle that also requires the interconnect bus. The remote Mainframe Extender must detect and resolve the deadlock condition by terminating the remote bus cycle by asserting BERR* on the remote bus. This will allow the bus cycle that owns the interconnect bus to complete successfully.

PERMISSION C.3:

Mainframe Extenders and interconnect buses **MAY** support non-interlocked bus arbitration.

OBSERVATION C.1:

Bus masters designed to operate in non-interlocked systems should accommodate the possibility of deadlock bus errors by retrying the bus cycle.

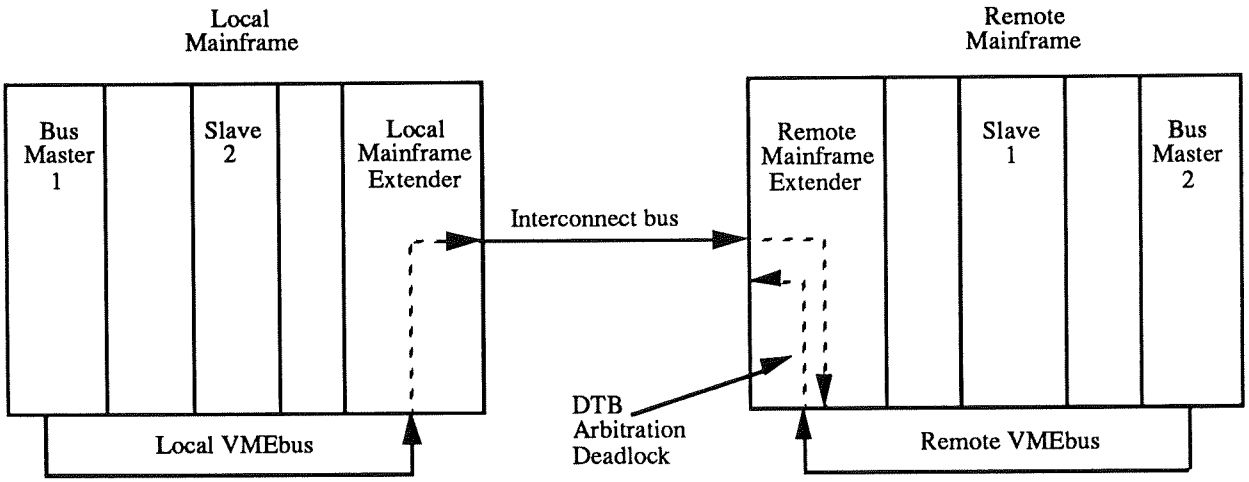


Figure C.3. Data Transfer Bus Arbitration Deadlock.

C.7 INTERRUPTS and INTERRUPT HANDLING between MAINFRAMES

Interrupts can be routed between mainframes using an optional interrupt routing mechanism. When an Interrupter asserts an interrupt line, the interrupt is routed across the appropriate interconnect bus(es) to the remote mainframe containing the Interrupt Handler.

The Interrupt Handler detects the interrupt and generates an interrupt acknowledge (IACK) cycle. The IACK signal propagates through the remote IACK daisy-chain until it reaches the Mainframe Extender, which intercepts the IACK signal and routes the IACK cycle to the (local) mainframe containing the Interrupter. The IACK signal then propagates down the local daisy-chain until it reaches the Interrupter. The Interrupter then responds with its Status/ID word.

RECOMMENDATION C.5:

Mainframe Extenders and interconnect buses that support the interrupt extension mechanism should support both inward and outward interrupt routing. Interrupt Handlers in both the local and remote mainframes should be able to detect and acknowledge interrupts generated by Interrupters in other mainframes.

OBSERVATION C.2:
Interrupt routing is unidirectional on each interrupt line.

The Resource Manager configures the interrupt routing mechanism in a manner compatible with the system requirements (see Section F.5, "Interrupt Resource Configuration").

OBSERVATION C.3:
Multiframe systems may have segmented interrupt lines. Each segment may have its own Interrupt Handler. For example, the IRQ2* lines in different mainframes may be isolated from each other, and may have different interrupt handlers.

C.8 TTL TRIGGERING between MAINFRAMES

TTL triggers can be routed between mainframes using an optional TTL trigger routing mechanism. When a TTL trigger source asserts a trigger line, the trigger is routed across the appropriate interconnect bus(es) to the remote mainframe containing the acceptor(s). The acceptor(s) respond according to the TTL trigger protocol in effect.

RULE C.5:
Mainframe Extenders and interconnect buses that support the TTL trigger routing mechanism **SHALL** support inward and outward TTL trigger routing. Acceptors in both the local and remote mainframes must be able to detect the assertion of TTL triggers by a source in the other mainframe.

OBSERVATION C.4:
TTL trigger routing is unidirectional on each TTL trigger level.

The Resource Manager may initially configure the TTL trigger resources in a manner consistent with the system requirements. The TTL trigger routing may be dynamically modified by each Mainframe Extender's commander.

OBSERVATION C.5:
Multiframe systems may have segmented TTL trigger lines. For example, the TTLTRG4* lines in different mainframes may be isolated from each other, and may be used simultaneously for different purposes.

OBSERVATION C.6:
TTLTRG protocol timings are not designed for operation in a multiframe system configuration. Mainframe Extender implementation-dependent specifications for alternate trigger source/acceptor timing requirements are likely to be critical to maintaining reliable operation of TTL trigger protocols in multiframe systems. Observation B.6.6 in VXI-1, Rev. 1.3 points out that the TTLTRG rise time in a fully loaded VXIbus mainframe can be up to 40 nsec. VXI-1, Rev. 1.3 Section B.6.2.3.1.2 allows for a 50 nsec TTLTRG pulse to shrink to 10 nsec (non-assertion polarity) within a single mainframe (these numbers have been verified by simulation). These effects become cumulative if the trigger signal passes through multiple mainframes. The signal degradation (pulse shrinkage in particular) associated with the Mainframe Extender implementation is also an important system consideration, and should be specified by Mainframe Extender vendors.

OBSERVATION C.7:
Aside from the pulse degradation effects, the TTL SYNC trigger, ASYNC trigger, and clock transmission protocols can be extended in multiframe systems by using the TTL trigger routing mechanism.

OBSERVATION C.8:
The TTL SEMI-SYNC protocol imposes timing constraints that will generally require local acceptor and protocol translation logic circuitry in each Mainframe Extender.

OBSERVATION C.9:
Extending the TTL STST and Data Transmission protocols requires a system level strategy for distributing the CLK10 and trigger/clock signals, and for controlling the skew between the clocks and the trigger/data signals. Synchronous operation of these protocols across multiple mainframes requires local generation of the clock and the trigger signals, with equal path lengths to all remote mainframes involved. This requirement potentially conflicts with other trigger and clock distribution strategies. Many design alternatives that optimize the different aspects are possible; the best choice is dictated by the system requirements.

C.9 ECL TRIGGERING between MAINFRAMES

ECL triggers can be routed between mainframes using an optional ECL trigger routing mechanism. When an ECL trigger source asserts a trigger line, the trigger is routed across the appropriate interconnect bus(es) to the remote mainframe containing the acceptor(s). The acceptor(s) respond according to the ECL trigger protocol in effect.

RULE C.6:
Mainframe Extenders and interconnect buses that support the ECL trigger routing mechanism **SHALL** support inward and outward ECL trigger routing. Acceptors in both the local and remote mainframes must be able to detect the assertion of ECL triggers by a source in the other mainframe.

OBSERVATION C.10:
ECL trigger routing is unidirectional on each ECL trigger level.

The Resource Manager may initially configure the ECL trigger resources in a manner consistent with the system requirements. The ECL trigger routing may be dynamically modified by each Mainframe Extender's commander.

OBSERVATION C.11:
Multiframe systems may have segmented ECL trigger lines. For example, the ECLTRG4 lines in different mainframes may be isolated from each other, and may used simultaneously for different purposes.

OBSERVATION C.12:
ECLTRG protocol timings are not designed for operation in a multiframe system configuration. Mainframe Extender implementation-dependent specifications for alternate trigger source/acceptor timing requirements are likely to be critical to maintaining reliable operation of ECL trigger protocols in multiframe systems. VXI-1, Rev. 1.3 Section B.6.2.4.1.1 allows for 2 nsec pulse shrinkage in the ECLTRG assertion or non-assertion time within a single mainframe (a 1 nsec shrinkage has been verified by simulation). This effect becomes cumulative if the trigger signal passes through multiple mainframes. The signal degradation (pulse shrinkage in particular) associated with the Mainframe Extender implementation is also an important system consideration, and should be specified by Mainframe Extender vendors.

OBSERVATION C.13:
Aside from the pulse degradation effects, the ECL SYNC trigger, ASYNC trigger, and clock transmission protocols can be extended in multiframe systems by using the ECL trigger routing mechanism.

OBSERVATION C.14:
The ECL SEMI-SYNC protocol imposes timing constraints that generally require local acceptor and protocol translation circuitry in each Mainframe Extender.

OBSERVATION C.15:
Extending the ECL ESTST and Data Transmission protocols requires a system level strategy for distributing the CLK100, SYNC100, and trigger/clock signals, and for controlling the skew between the clocks and the trigger/data signals. Synchronous operation of these protocols across multiple mainframes requires local generation of the clock and the trigger signals, with equal path lengths to all remote mainframes involved. This requirement potentially conflicts with other trigger and clock distribution strategies. Many design alternatives that optimize the different aspects are possible; the best choice is dictated by the system requirements.

OBSERVATION C.16:
Neither this document nor VXI-1, Rev 1.3 define a standard implementation for extending ECL signals outside a mainframe.

C.10 UTILITY BUS OPERATION between MAINFRAMES

The signals ACFAIL*, SYSFAIL*, and SYSRESET* can be routed between mainframes using an optional utility bus routing mechanism. When a local device asserts one of these utility bus signals, the signal is routed across the appropriate interconnect bus(es) to the remote mainframe(s).

Unlike the interrupt and trigger extension mechanisms, each utility bus signal's input and output routing is independently configurable at each Mainframe Extender. This capability permits fully bidirectional connection of the utility bus, either globally or within a specific group of VXIbus subsystems.

RULE C.7:
Mainframe Extenders and interconnect buses that support the utility bus routing mechanism **SHALL** support fully bidirectional routing of each signal. Devices in both the local and remote mainframes must be able to detect the assertion of the utility bus signals by devices in the other mainframe.

OBSERVATION C.17:
Special circuitry is required in each Mainframe Extender to support fully bidirectional routing of the utility bus signals without "locking up" the utility bus.

The Resource Manager configures the Utility bus resources in a manner consistent with the system requirements.

D. MAINFRAME EXTENDER:

This section defines the Mainframe Extender characteristics and register set.

A Mainframe Extender is a reserved subclass extended device.

RULE D.1:
A Mainframe Extender **SHALL** be a static configuration (SC) device.

RECOMMENDATION D.1:
A Mainframe Extender should have Slot 0 device capability.

RECOMMENDATION D.2:
A Mainframe Extender's Slot 0 functionality should be defeatable so that it can reside in any VXIbus mainframe slot.

D.1 MAINFRAME EXTENDER REGISTERS

The Mainframe Extender register map is shown in Figure D.1.

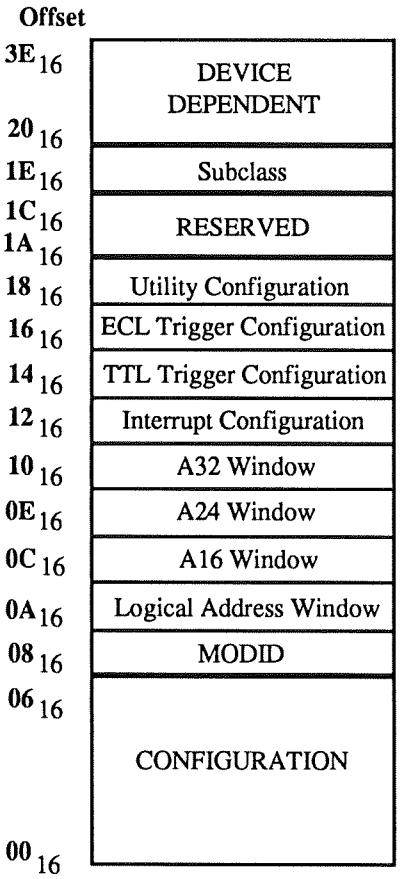


Figure D.1. Mainframe Extender Register Map.

RULE D.2:
A Mainframe Extender **SHALL** implement D16 and D08(E0) slave capability to the registers defined in this section.

RULE D.3:
Access to the Mainframe Extender's registers from the interconnect bus and the VXIbus **SHALL** always be enabled.

OBSERVATION D.1:
The Resource Manager could not initialize or control a Mainframe Extender if it could not access its registers.

D.1.1 Configuration Registers

The configuration registers have the same definition and function for Mainframe Extender devices as they do for other VXIbus devices. The functionality and content of these registers are defined in VXI-1, Rev. 1.3, with the following additional Status Register definition:

Status Register: This register (at offset 04h) provides information about the Mainframe Extender's status. Bit 8 is defined as follows:

Bit #	15	14	13 ← 9	8	7 ← 4	3	2	1 ← 0
Contents	A24/A32 Active	MODID*	Device Dependent	ACCDIR	Device Dependent	Ready	Passed	Device Dependent

- ACCDIR (Access Direction): A zero indicates that the current access to the Status register is originating from the VXIbus side. A one indicates that the access originated from the interconnect bus side.

OBSERVATION D.2:
VXIbus devices can use the ACCDIR bit to distinguish between parent-side and child-side Mainframe Extenders. This also allows them to determine the correct sense of the direction bits when modifying or interpreting the contents of the Mainframe Extender registers.

D.1.2 MODID Register

This register is required for Slot 0 Mainframe Extenders. It allows a Resource Manager to drive the VXIbus MODID lines and read their status. The MODID register is identical in format and functionality to the Register-Based Slot 0 device MODID register defined in VXI-1, Rev. 1.3, Section C.4.3.1.1.

D.1.3 Logical Address Window Register

This register controls the range of Logical Addresses that are routed between local and remote devices. It also controls the direction (inward or outward) to which the range applies, and whether the Logical Address routing function is enabled.

The fields of the Logical Address Window register are defined below:

Bit #	15	14	13	12	11	10 ← 8	7 ← 0
Contents	0	LAEN	LADIR	1	1	LASIZE	LABASE

- Bits 15, 12, and 11 are read-only. Writes to these bits have no effect.

The following fields are readable and writable:

- LAEN (Logical Address Enable): A one indicates that the Logical Address window is enabled. A zero indicates that the window is disabled.

- LADIR (Logical Address Direction): When this bit is zero, the Logical Address window configuration applies to outward bus cycles. When this bit is one, the window applies to inward bus cycles.

  When LAEN is one, the Mainframe Extender routes the Logical Address range defined by LASIZE and LABASE in the direction indicated by LADIR. It also routes the *complement* of the defined address range in the *opposite* direction. In either case, it detects cycles directed to its own Logical Address, and does not route them any further.

- LASIZE (Logical Address Size): The content of this field is the number of significant address bits in the LABASE field. The number of Logical Addresses in the window is given by

  2^{8-i}

  where *i* is the value of the LASIZE field.

OBSERVATION D.3:
Because *i* can have a range from 0 to 7, the minimum size of an enabled Logical Address window is 2, and the maximum is 256 (100_{16}).

- LABASE (Logical Address Base): This field defines the base address of the Mainframe Extender's Logical Address window. The *i* most significant bits of the LABASE field are the *i* most significant bits of the base Logical Address, where *i* is the value in the LASIZE field. The 8-*i* least significant bits are meaningless. The base address is derived by replacing these bits with zeros.

D.1.4 A16 Window Register

This register controls the range of addresses in the lower 48 kilobytes of A16 space that are routed between local and remote devices. It also controls the direction (inward or outward) to which the range applies, and whether the A16 routing function is enabled.

OBSERVATION D.4:

The A16 window accommodates non-VXI devices that must reside in A16 space.

The fields of the A16 Window register are defined below:

Bit #	15	14	13	12	11	10 ← 8	7 ← 0
Contents	0	A16EN	A16DIR	1	1	A16SIZE	A16BASE

- Bits 15, 12, and 11 are read-only. Writes to these bits have no effect.

The following fields are readable and writable:

- A16EN (A16 Enable): A one indicates that the A16 window is enabled. A zero indicates that the window is disabled.

- A16DIR (A16 Direction): When this bit is zero, the A16 window configuration applies to outward bus cycles. When it is one, the window configuration applies to inward bus cycles.

  When A16EN is one, the Mainframe Extender routes the A16 address range defined by A16BASE and A16SIZE in the direction indicated by A16DIR. It also routes the *complement* of the defined address range in the *opposite* direction.

- A16SIZE (A16 Size): The content of this field is the number of significant address bits in the A16BASE field. The number of A16 addresses in the window is given by

  $$256 * 2^{8-i}$$

  where *i* is the value of the A16SIZE field.

RULE D.4:

The A16 window routing function **SHALL NOT** route A16 addresses above 49,151 ($BFFF_{16}$) into or out of the mainframe.

OBSERVATION D.5:

A16 addresses above 49,151 are reserved for VXIbus Logical Address space.

OBSERVATION D.6:

Because *i* can have a range from 0 to 7, the minimum size of an enabled A16 window is 512 bytes (200_{16}). The window size for i=1 is 32 kilobytes (8000_{16}). The window size for i=0 is 48 kilobytes ($C000_{16}$).

- A16BASE (A16 base): This field defines the base address of the Mainframe Extender's A16 window. The *i* most significant bits of the A16BASE field are the *i* most significant bits of the base A16 address, where *i* is the value of the A16SIZE field. The 8-*i* least significant bits are meaningless. The base address is derived by replacing these bits with zeros and multiplying the result by 256 (100_{16}).

D.1.5 A24 Window Register

This register controls the range of addresses in A24 space that are routed between local and remote devices. It also controls the direction (inward or outward) to which the range applies, and whether the A24 routing function is enabled.

The fields of the A24 Window register are defined below:

Bit #	15	14	13	12	11	10 ← 8	7 ← 0
Contents	0	A24EN	A24DIR	1	1	A24SIZE	A24BASE

- Bits 15, 12, and 11 are read-only. Writes to these bits have no effect.

The following fields are readable and writable:

- A24EN (A24 Enable): A one indicates that the A24 window is enabled. A zero indicates that the window is disabled.

- A24DIR (A24 Direction): When this bit is zero, the A24 window configuration applies to outward bus cycles. When it is one, the window configuration applies to inward bus cycles.

  When A24EN is one, the Mainframe Extender routes the A24 address range defined by A24BASE and A24SIZE in the direction indicated by A24DIR. It also routes the *complement* of the defined address range in the *opposite* direction.

- A24SIZE (A24 Size): The content of this field is the number of significant address bits in the A24BASE field. The number of A24 addresses in the window is given by

  $$65536 * 2^{8-i}$$

  where i is the value of the A24SIZE field.

  OBSERVATION D.7:
  Because i can have a range from 0 to 7, the minimum size of an enabled A24 window is 128 kilobytes (20000_{16}), and the maximum is 16 megabytes (1000000_{16}).

- A24BASE (A24 base): This field defines the base address of the Mainframe Extender's A24 window. The i most significant bits of the A24BASE field are the i most significant bits of the base A24 address, where i is the value of the A24SIZE field. The $8-i$ least significant bits are meaningless. The base address is given by replacing these bits with zeros and multiplying the result by 65,536 (10000_{16}).

D.1.6 A32 Window Register

This optional register controls the range of addresses in A32 space that are routed between local and remote devices. It also controls the direction (inward or outward) to which the range applies, and whether the A32 routing function is enabled.

The fields of the A32 Window register are defined below:

Bit #	15	14	13	12	11	10 ← 8	7 ← 0
Contents	0	A32EN	A32DIR	1	1	A32SIZE	A32BASE

- Bits 15, 12, and 11 are read-only. Writes to these bits have no effect.

The following fields are readable and writable.

- A32EN (A32 Enable): A one indicates that the A32 window is enabled. A zero indicates that the window is disabled.

- A32DIR (A32 Direction): When this bit is zero, the A32 window configuration applies to outward bus cycles. When it is one, the window configuration applies to inward bus cycles.

  When A32EN is one, the Mainframe Extender routes the A32 address range defined by A32BASE and A32SIZE in the direction indicated by A32DIR. It also routes the *complement* of the defined address range in the *opposite* direction.

- A32SIZE (A32 Size): The content of this field is the number of significant address bits in the A32BASE field. The number of A32 addresses in the window is given by

  $$16{,}777{,}216 * 2^{8-i}$$

  where *i* is the value of the A32SIZE field.

  OBSERVATION D.8:
  Because *i* can have a range from 0 to 7, the minimum size of an enabled A32 window is 32 megabytes (2000000_{16}), and the maximum is 4 gigabytes (100000000_{16}).

- A32BASE (A32 base): This field defines the base address of the Mainframe Extender's A32 window. The *i* most significant bits of the A32BASE field are the *i* most significant bits of the base A32 address, where *i* is the value of the A32SIZE field. The 8-*i* least significant bits are meaningless. The base address is given by replacing these bits with zeros and multiplying the result by 16,777,216 (1000000_{16}).

If the Mainframe Extender does not support A32 bus cycle routing, writes to the A32 Window register have no effect, and reads return $FFFF_{16}$.

D.1.7 Interrupt Configuration Register

This optional register controls which VMEbus interrupt lines are routed between local and remote devices.

RULE D.5:

IF a Mainframe Extender supports the interrupt configuration register,
THEN it **MUST** support seven interrupt lines on the interconnect bus, which are mapped one-to-one with the seven VMEbus interrupt lines.

The fields of the Interrupt Configuration register are defined below:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Contents	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1
		INTEN								INTDIR						

- Bits 15 and 7 are read-only. If the register is supported, a read of these bits returns zeros. Writes to these bits have no effect.

- INTEN (Interrupt Enable): Bits 8 to 14 correspond to Interrupt Request lines 1 to 7, respectively. A one in any bit position indicates that the corresponding VMEbus Interrupt Request line is routed to the corresponding interconnect bus Interrupt Request line, in the direction indicated by the corresponding bit in the INTDIR field. A zero indicates that interrupt routing on that Interrupt Request line is disabled. INTEN is readable and writable.

- INTDIR (Interrupt Direction): Bits 0 to 6 correspond to Interrupt Request lines 1 to 7, respectively. A one in any bit position indicates that the corresponding interconnect bus Interrupt Request line is routed into the corresponding VMEbus Interrupt Request line. A zero indicates that the corresponding VMEbus Interrupt Request line is routed out to the corresponding interconnect bus Interrupt Request line. Each Interrupt Request line routing is enabled by the corresponding bit in the INTEN field. INTDIR is readable and writable.

If the Mainframe Extender does not support interrupt extensions, writes to the Interrupt Configuration register have no effect, and reads return $FFFF_{16}$.

In addition to routing IRQ lines, the Mainframe Extender also routes the corresponding interrupt acknowledge (IACK) cycles in the complementary direction.

RULE D.6:

When a Mainframe Extender detects an IACK cycle on the local bus for an IRQ that it is asserting on behalf of a remote Interrupter, it **SHALL** intercept the IACK daisy-chain on the local bus in the same manner as a local Interrupter, and route the IACK cycle onto the interconnect bus.

RULE D.7:

When a Mainframe Extender detects an IACK cycle on the interconnect bus for an IRQ that it is asserting on the interconnect bus, it **SHALL** route the IACK cycle onto the local bus.

D.1.8 TTL Trigger Configuration Register

This optional register controls which TTL trigger lines are routed between local and remote devices. Support for this register is indicated in the TTL* bit in the Utility Configuration Register.

RULE D.8:

IF a Mainframe Extender supports the TTL Trigger Configuration register,
THEN it **MUST** support eight trigger lines on the interconnect bus, which are mapped one-to-one with the eight VXIbus TTL trigger lines.

The fields of the TTL Trigger Configuration register are defined below:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Contents	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	TTLTRGEN								TTLTRGDIR							

- TTLTRIGEN (TTL Trigger Enable): Bits 8 to 15 correspond to VXIbus TTL trigger lines 0 to 7, respectively. A one in any bit position indicates that the corresponding VXIbus TTL trigger line is routed to the corresponding interconnect bus trigger line, in the direction indicated by the corresponding bit in the TTLTRGDIR field. A zero indicates that trigger mapping is disabled for that trigger line. TTLTRIGEN is readable and writable.

- TTLTRIGDIR (TTL Trigger Direction): Bits 0 to 7 correspond to VXIbus TTL trigger lines 0 to 7, respectively. A one in any bit position indicates that the corresponding interconnect bus trigger line is routed into the corresponding VXIbus TTL trigger line. A zero indicates that the corresponding VXIbus TTL trigger line is routed out to the corresponding interconnect bus trigger line. The routing is individually enabled for each trigger line by the corresponding bit in the TTLTRIGEN field. TTLTRIGDIR is readable and writable.

OBSERVATION D.9:

The trigger extension supported by the TTL Trigger Configuration Register is a simple routing mechanism. Any TTL trigger protocol translation configuration should be defined in the Device Dependent Registers.

If the Mainframe Extender does not support TTL trigger extensions, writes to the TTL Trigger Configuration register have no effect, and reads return $FFFF_{16}$.

D.1.9 ECL Trigger Configuration Register

This optional register controls which ECL trigger lines are routed between local and remote devices. Support for this register is indicated in the ECL2* and ECL3* bits in the Utility Configuration Register.

RULE D.9:

IF a Mainframe Extender supports the ECL Trigger Configuration register,
THEN it **MUST** support either two trigger lines on the interconnect bus that are mapped one-to-one with the VXIbus P2 ECL trigger lines, or six trigger lines on the interconnect bus that are mapped one-to-one with the VXIbus P2 and P3 ECL trigger lines.

The fields of the ECL Trigger Configuration register are defined below:

Bit #	15 ← 14	13	12	11	10	9	8	7 ← 6	5	4	3	2	1	0
Contents		5	4	3	2	1	0		5	4	3	2	1	0
	RESERVED	ECLTRGEN						RESERVED	ECLTRGDIR					

- ECLTRIGEN (ECL Trigger Enable): Bits 8 to 13 correspond to VXIbus ECL trigger lines 0 to 5, respectively. A one in any bit position indicates that the corresponding VXIbus ECL trigger line is routed to the corresponding interconnect bus trigger line, in the direction indicated by the corresponding bit in the ECLTRGDIR field. A zero indicates that trigger mapping is disabled for that trigger line. ECLTRIGEN is readable and writable.

- ECLTRIGDIR (ECL Trigger Direction): Bits 0 to 5 correspond to VXIbus ECL trigger lines 0 to 5, respectively. A one in any bit position indicates that the corresponding interconnect bus trigger line is routed into the corresponding VXIbus ECL trigger line. A zero indicates that the corresponding VXIbus ECL trigger line is routed out to the corresponding interconnect bus trigger line. The routing is individually enabled for each trigger line by the corresponding bit in the ECLTRIGEN field. ECLTRIGDIR is readable and writable.

- RESERVED: The read and write contents of Bits 6, 7, 14, and 15 are reserved for future definition. Until defined, these bits should always be written with ones, and reads of the reserved bits will return ones.

OBSERVATION D.10:

The ECL trigger extension supported by the ECL Trigger Configuration register is a simple routing mechanism. Any ECL trigger protocol translation configuration should be defined in the Device Dependent Registers.

If the Mainframe Extender does not support ECL trigger extensions, writes to the ECL Trigger Configuration register have no effect, and a read returns $FFFF_{16}$.

If the Mainframe Extender supports only P2 ECL trigger extension, writes to bit positions 2 to 5 and 10 to 13 have no effects, and reads return ones in these bit positions.

D.1.10 Utility Configuration Register

This register controls which utility bus lines (ACFAIL*, SYSFAIL*, and SYSRESET*) are routed between local and remote devices, and indicates support of the TTL trigger, ECL trigger, and utility bus routing mechanisms.

RULE D.10:

IF a Mainframe Extender supports the utility bus routing mechanism,
THEN it **MUST** support three utility bus lines on the interconnect bus, which are mapped one-to-one with the three utility bus lines ACFAIL*, SYSFAIL*, and SYSRESET*.

The fields of the Utility Configuration register are defined below:

Bit #	15	14	13	12	11 ← 6	5	4	3	2	1	0
Contents	TTL*	ECL3*	ECL2*	UTIL*	RESERVED	ACFIN	ACFOUT	SFIN	SFOUT	SRIN	SROUT

- RESERVED: The read and write contents of this field are reserved for future definition. Until defined, these bits should always be written with ones, and reads of these bits will return ones.

The following fields are read-only. Writes to these bits have no effect:

- TTL*: A zero in this bit position indicates that the Mainframe Extender supports routing of the TTL trigger lines (TTLTRG0-7*) as defined by the TTL Trigger Configuration Register.
- ECL3*: A zero in this bit position indicates that the Mainframe Extender supports the routing of the P3 ECL trigger lines (ECLTRG2-5), as defined by the ECL Trigger Configuration Register.
- ECL2*: A zero in this bit position indicates that the Mainframe Extender supports the routing of the P2 ECL trigger lines (ECLTRG0-1), as defined by the ECL Trigger Configuration Register.
- UTIL*: A zero in this bit position indicates that the Mainframe Extender supports the routing of the utility bus signals ACFAIL*, SYSFAIL* and SYSRESET*, as defined by bits 0 to 5 of this register.

The following fields are readable and writable:

- ACFIN (ACFAIL* In): A one in this bit position indicates that the interconnect bus ACFAIL* line is routed in to the VMEbus ACFAIL* line. A zero indicates that ACFAIL* inward routing is disabled. ACFIN is readable and writable.
- ACFOUT (ACFAIL* Out): A one in this bit position indicates that the VMEbus ACFAIL* line is routed out to the interconnect bus ACFAIL* line. A zero indicates that ACFAIL* outward routing is disabled. ACFOUT is readable and writable.
- SFIN (SYSFAIL* In): A one in this bit position indicates that the interconnect bus SYSFAIL* line is routed in to the VMEbus SYSFAIL* line. A zero indicates that SYSFAIL* inward routing is disabled. SFIN is readable and writable.
- SFOUT (SYSFAIL* Out): A one in this bit position indicates that the VMEbus SYSFAIL* line is routed out to the interconnect bus SYSFAIL* line. A zero indicates that SYSFAIL* outward routing is disabled. SFOUT is readable and writable.

- SRIN (SYSRESET* In): A one in this bit position indicates that the interconnect bus SYSRESET* line is routed in to the VMEbus SYSRESET* line. A zero indicates that SYSRESET* inward routing is disabled. SRIN is readable and writable.

- SROUT (SYSRESET* Out): A one in this bit position indicates that the VMEbus SYSRESET* line is routed out to the interconnect bus SYSRESET* line. A zero indicates that SYSRESET* outward routing is disabled. SROUT is readable and writable.

If the Mainframe Extender does not support utility extensions, writes to the Utility Configuration register have no effect, and reads of bits 0 to 5 return all ones.

D.1.11 Reserved Registers

These registers are reserved for future definition. Until defined, the read contents of these 16-bit registers is $FFFF_{16}$. The effect of writes to these registers is undefined.

D.1.12 Subclass Register

The read content of this 16-bit register is $FFFC_{16}$.

The effect of writes to this register is undefined.

D.1.13 Device Dependent Registers

These registers are available for proprietary implementations of additional extension capabilities or for other device-dependent functionality.

D.2 INITIALIZATION and CONTROL

OBSERVATION D.11:
As required by VXI-1, Rev. 1.3, Rule C.4.18, a Slot 0 Mainframe Extender clears the Output Enable bit in its MODID register to zero while it is in the HARD RESET or SOFT RESET states.

RULE D.11:
A Mainframe Extender **SHALL** clear the LAEN field in its Logical Address Window register to zero while it is in the HARD RESET state.

RULE D.12:
A Mainframe Extender **SHALL** clear the A16EN field in its A16 Window register to zero while it is in the HARD RESET state.

RULE D.13:
A Mainframe Extender **SHALL** clear the A24EN field in its A24 Window register to zero while it is in the HARD RESET state.

RULE D.14:
A Mainframe Extender **SHALL** clear the A32EN field in its A32 Window register to zero while it is in the HARD RESET state.

RULE D.15:
A Mainframe Extender **SHALL** clear the contents of the Interrupt Configuration, TTL Trigger Configuration, and ECL Trigger Configuration registers to zero while it is in the HARD RESET state.

RULE D.16:
A Mainframe Extender **SHALL** clear the contents of the Utility Configuration register to zero when the mainframe is powered-on.

RULE D.17:
The contents of the Utility Configuration register fields **SHALL NOT** be affected in the HARD RESET state.

RULE D.18:
The contents of the Logical Address Window, A16 Window, A24 Window, A32 Window, Interrupt Configuration, TTL Trigger Configuration, ECL Trigger Configuration, and Utility Configuration registers **SHALL NOT** be affected in the SOFT RESET state.

The system memory map and interrupt resources are controlled by the Resource Manager. The trigger routing is initially configured by the Resource Manager, and then dynamically modified by each Mainframe Extender's commander.

Only the Resource Manager is allowed to write to a Mainframe Extender's Logical Address Window, A16 Window, A24 Window, A32 Window, Interrupt Configuration, and Utility Configuration registers.

RULE D.19:
A Mainframe Extender's commander **SHALL NOT** write to the Logical Address Window, A16 Window, A24 Window, A32 Window, Interrupt Configuration, and Utility Configuration registers, unless the commander is also the Resource Manager.

PERMISSION D.1:
Any device **MAY** read a Mainframe Extender's registers.

OBSERVATION D.12:
At power-on, the mainframes are isolated, and the Resource Manager brings them into the system one at a time. The address window and interrupt/trigger/utility bus routing configurations are retained while the Mainframe Extender is in the SOFT RESET state.

E. SYSTEM CONFIGURATION:

This section describes the rules and recommendations governing the multiframe system configuration task and the minimum Resource Manager tools available to the system integrator.

E.1 LOGICAL ADDRESS WINDOW CONFIGURATION

A bus's Logical Address range is the range of Logical Addresses occupied by devices on the bus and its descendant buses.

The system integrator is responsible for configuring the Logical Address of each device in the system such that the following rules are obeyed:

RULE E.1:
Each bus's Logical Address range **SHALL NOT** contain the Logical Address range of any other bus, except its descendant buses.

RULE E.2:
Each bus's Logical Address range **SHALL** contain the Logical Address ranges of all of its descendant buses.

RULE E.3:
A bus's Logical Address range **SHALL NOT** include the Logical Address of any device on any of its ancestor buses.

PERMISSION E.1:
A Mainframe Extender 's Logical Address **MAY** be located in the Logical Address range of its descendant bus.

OBSERVATION E.1:
These rules allow the Resource Manager to configure the system Logical Address map in the structured manner required for transparent communication between all devices.

A bus's Logical Address range can be described in terms of its *base address* and its *span* (the number of Logical Addresses in the range). This representation is used to derive the Logical Address range of all buses, according to the following rules:

1. The address span of each bus is the least power of 2 that is greater than or equal to the sum of its children's spans and the number of devices on the bus.

2. The root bus's Logical Address span is 256 (100_{16}).

3. The Logical Address window size of each extender is the same as the Logical Address span of the bus connected to its downward side.

E.1.1 System Configuration Example

This example is not intended to be exhaustive or definitive. It demonstrates one methodology for planning a system Logical Address map.

The following additional configuration guidelines are used in this example to determine the Logical Addresses assigned to the devices in the system:

1. In each mainframe, the parent-side Mainframe Extender's Logical Address is the same as its Logical Address window base.

2. In each mainframe, the Logical Address windows of all child-side extenders are located at the highest addresses of that mainframe's range.

3. In each mainframe, all child-side extenders have the highest Logical Addresses below the locations of the child-side extender Logical Address windows.

4. On each interconnect bus, the Logical Address windows of parent-side extenders and the Logical Addresses of stand-alone devices are allocated starting at the highest available addresses within the bus's address range, beginning with the largest window or Logical Address block.

In this example, we will consider the case in which a system integrator needs to plan the Logical Address map for a system consisting of a root mainframe connected to three mainframes and two VXIbus devices by one interconnect bus, and to a fourth mainframe by a second interconnect bus. One of the mainframes on the first interconnect bus is connected by a third interconnect bus to a fifth mainframe (see Figure E.1).

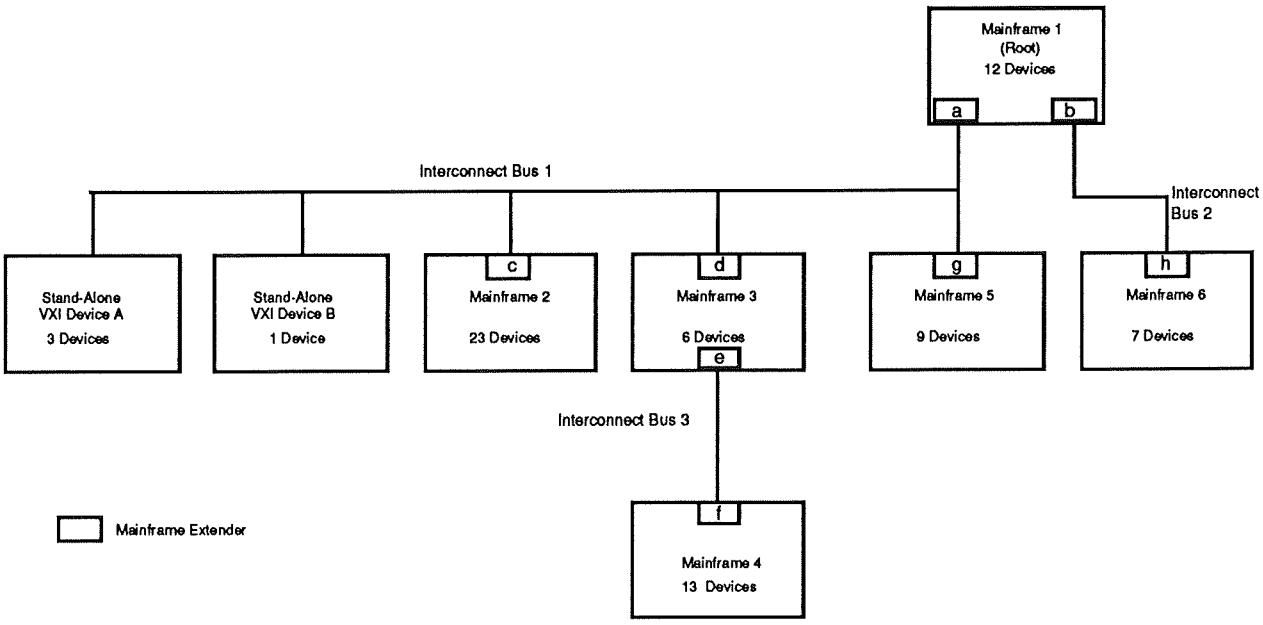


Figure E.1. Example System Topology.

In this example, assume that the number of Logical Addresses occupied by local devices in each mainframe is known *a priori* to be as shown in Figure E.1.

Each bus's Logical Address span can be determined by starting at the lowest level of the tree and working up each branch as follows:

1. Mainframe 4 requires 13 Logical Addresses. Because mainframe Logical Address spans are allocated by powers of 2, Mainframe 4 's Logical Address span is 16 (10_{16}).

2. Interconnect Bus 3 requires no Logical Addresses. Its Logical Address span must contain Mainframe 4's Logical Address span, so its span is 16 (10_{16}).

3. Mainframe 3 requires 6 Logical Addresses. Its Logical Address span must also contain Interconnect Bus 3's Logical Address span, so Mainframe 3's Logical Address span is 6 + 16 = 22 –> 32 (20_{16}).

4. Mainframe 2's Logical Address span is 23 –> 32 (20_{16}).

5. Mainframe 5's Logical Address span is 9 –> 16 (10_{16}).

6. Interconnect Bus 1 requires 4 Logical Addresses for the stand-alone devices. Its Logical Address span must also contain the Logical Address spans for Mainframes 2, 3, and 5, so its span is 4 + 32 + 32 + 16 = 84 -> 128 (80_{16}).

7. Mainframe 6's Logical Address span is 7 –> 8 (8_{16}).

8. Interconnect Bus 2 requires no Logical Addresses. Its Logical Address span must contain Mainframe 6's Logical Address span, so its span is 8 (8_{16}).

9. The root mainframe's Logical Address span is always 256, so Mainframe 1's span is 256 (100_{16}).

These results are summarized in Table E.1.

Node	Number of Logical Addresses Occupied by Local Devices	Span
Mainframe 1	12	100_{16}
Mainframe 2	23	20_{16}
Mainframe 3	6	20_{16}
Mainframe 4	13	10_{16}
Mainframe 5	9	10_{16}
Mainframe 6	7	8_{16}
Interconnect Bus 1	4	80_{16}
Interconnect Bus 2	0	8_{16}
Interconnect Bus 3	0	10_{16}

Table E.1. Logical Address Span Example.

The Logical Address window size of each Mainframe Extender is the same as the Logical Address span of its descendant bus. The Logical Address window sizes for each Mainframe Extender can be taken directly from Table E.1.

Each Mainframe Extender's Logical Address window base can be determined by starting at the root and working down each branch of the tree, assigning the highest available addresses to the Mainframe Extender with the largest window size, as follows:

1. Mainframe Extender **a** has the largest window size in Mainframe 1. Assign its base to the highest available 128-address boundary, 80_{16}. Notice that only 127 Logical Addresses are available in this window, because FF_{16} is reserved for DC devices.

2. Assign the window bases for parent-side Mainframe Extenders on Interconnect Bus 1. Mainframe Extenders **c** and **d** both have window sizes of 32. Assign **c**'s base to $E0_{16}$, and **d**'s base to $C0_{16}$. Assign **g**'s base to $B0_{16}$.

3. Assign the window bases for Mainframe Extenders on Interconnect Bus 3. Assign the bases of Mainframe Extenders **e** and **f** to $D0_{16}$.

4. Assign the window bases for Mainframe Extenders on Interconnect Bus 2. Assign the bases of Mainframe Extenders **b** and **h** to 78_{16}.

Each Mainframe Extender's Logical Address can be determined as follows:

1. A parent-side Mainframe Extender's Logical Address is the same as its Logical Address window base, so extenders **c**, **d**, **f**, **g**, and **h** have Logical Addresses $E0_{16}$, $C0_{16}$, $D0_{16}$, $B0_{16}$, and 78_{16}, respectively.

2. A child-side Mainframe Extender's Logical Address is the highest available address in the mainframe in which it resides. When two or more child-side extenders reside in one mainframe, the one with the highest window base is assigned the highest Logical Address. Mainframe Extender **a** has Logical Address 77_{16}, **b** has Logical Address 76_{16}, and **e** has Logical Address CF_{16}.

These results are summarized in Table E.2.

Mainframe Extender	Logical Address Window Size	Logical Address Window Base	Logical Address
a	80_{16}	80_{16}	77_{16}
b	8_{16}	78_{16}	76_{16}
c	20_{16}	$E0_{16}$	$E0_{16}$
d	20_{16}	$C0_{16}$	$C0_{16}$
e	10_{16}	$D0_{16}$	CF_{16}
f	10_{16}	$D0_{16}$	$D0_{16}$
g	10_{16}	$B0_{16}$	$B0_{16}$
h	8_{16}	78_{16}	78_{16}

Table E.2. Logical Address Window Configuration Example.

The base of each mainframe's Logical Address range is the same as its parent-side extender's Logical Address window base. Its range includes the number of Logical Addresses in its span. The root mainframe's Logical Address range is always 0_{16} to FF_{16}. The local devices in each mainframe are located at Logical Addresses between the base and the child-side extender Logical Address(es).

The base of each interconnect bus's Logical Address range is the same as its child-side extender's Logical Address window base. Its range is the child-side extender's Logical Address window size. Stand-alone devices are located at Logical Addresses between the interconnect bus's Logical Address base and the Logical Address windows of parent-side extenders connected to the bus. In the case of Interconnect Bus 1, Devices A and B can be located anywhere between Logical Address 80_{16} and AF_{16}.

These results are summarized in Table E.3 and shown in Figure E.2.

Node	Logical Address Base	Logical Address Span	Range	Local Device Addresses	Child-Side Extender Logical Address
Mainframe 1	0_{16}	100_{16}	0_{16} to FF_{16}	0_{16} to 77_{16}	76_{16}, 77_{16}
Mainframe 2	$E0_{16}$	20_{16}	$E0_{16}$ to FF_{16}	$E0_{16}$ to FF_{16}	
Mainframe 3	$C0_{16}$	20_{16}	$C0_{16}$ to DF_{16}	$C0_{16}$ to CF_{16}	CF_{16}
Mainframe 4	$D0_{16}$	10_{16}	$D0_{16}$ to DF_{16}	$D0_{16}$ to DF_{16}	
Mainframe 5	$B0_{16}$	10_{16}	$B0_{16}$ to BF_{16}	$B0_{16}$ to BF_{16}	
Mainframe 6	78_{16}	8_{16}	78_{16} to $7F_{16}$	78_{16} to $7F_{16}$	
Interconnect Bus 1	80_{16}	80_{16}	80_{16} to FF_{16}	80_{16} to AF_{16}	76_{16}
Interconnect Bus 2	78_{16}	8_{16}	78_{16} to $7F_{16}$		77_{16}
Interconnect Bus 2	$D0_{16}$	10_{16}	$D0_{16}$ to DF_{16}		CF_{16}

Table E.3. Logical Address Range Example.

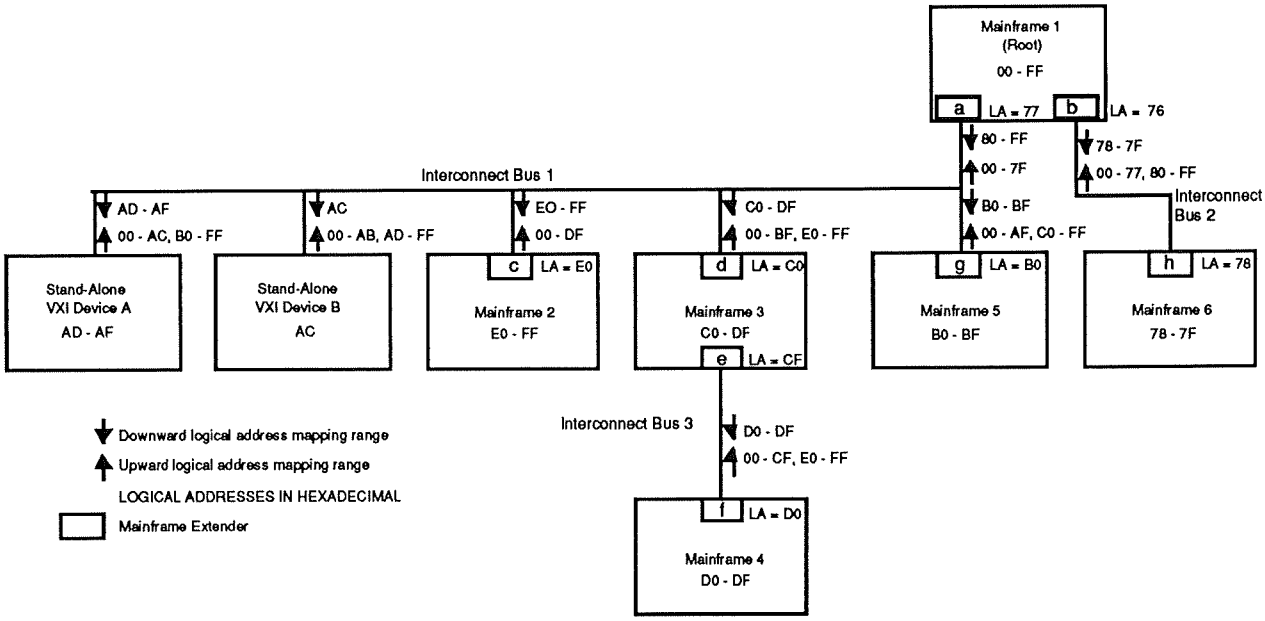


Figure E.2 Example Logical Address Map.

E.2 A16 WINDOW CONFIGURATION

A bus's A16 address range is the range of A16 address space occupied by devices on the bus and its descendant buses.

The system integrator is responsible for configuring the A16 address range for each bus according to the following rules:

RULE E.4:
Each mainframe's A16 address range **SHALL NOT** contain the A16 address range of any other bus, except its descendants.

RULE E.5:
Each bus's A16 address range **SHALL** contain the A16 address ranges of all of its descendant buses.

RULE E.6:
A bus's A16 address range **SHALL NOT** extend above $BFFF_{16}$.

OBSERVATION E.2:
A16 addresses above $BFFF_{16}$ are reserved for Logical Address space.

OBSERVATION E.3:
The A16 windows are not normally used in systems consisting entirely of VXIbus devices.

RECOMMENDATION E.1:
A multiframe Resource Manager should provide some method to configure the A16 windows.

E.3 A24 WINDOW CONFIGURATION

A bus's A24 address range is the range of A24 address space occupied by devices on the bus and its descendant buses.

The system integrator is responsible for configuring the A24 address range for each bus according to the following rules:

RULE E.7:
Each bus's A24 address range **SHALL NOT** contain the A24 address range of any other bus, except its descendants.

RULE E.8:
Each bus's A24 address range **SHALL** contain the A24 address ranges of all of its descendant buses.

OBSERVATION E.4:
The Resource Manager will auto-configure the A24 address range of each bus in a system consisting entirely of VXIbus devices.

E.4 A32 WINDOW CONFIGURATION

A bus's A32 address range is the range of A32 address space occupied by devices on the bus and its descendant buses.

The system integrator is responsible for configuring the A32 address range for each bus according to the following rules:

RULE E.9:
Each bus's A32 address range **SHALL NOT** contain the A32 address range of any other bus, except its descendants.

RULE E.10:
Each bus's A32 address range **SHALL** contain the A32 address range of all of its descendant buses.

OBSERVATION E.5:
The Resource Manager will auto-configure the A32 address range of each bus in a system consisting entirely of VXIbus devices.

E.5 INTERRUPT CONFIGURATION

The system integrator is responsible for determining the configuration of the system interrupt resources.

OBSERVATION E.6:
VXI-1, Rev. 1.3 requires a Resource Manager to provide a means for the system integrator to supply interrupt resource configuration information.

E.6 TRIGGER CONFIGURATION

The system integrator is responsible for determining the configuration of the system trigger resources.

OBSERVATION E.7:

It is recommended that a multiframe Resource Manager provide a means for the system integrator to supply TTL and ECL trigger resource configuration information.

E.7 UTILITY CONFIGURATION

The system integrator is responsible for determining the configuration of the system utility resources.

OBSERVATION E.8:

It is recommended that a multiframe Resource Manager provide a means for the system integrator to supply utility bus configuration information.

F. MULTIFRAME RESOURCE MANAGER OPERATION:

This section describes the algorithms by which the Resource Manager can:

- Configure the Logical Address map
- Identify the devices in the system
- Manage the system self-tests
- Configure the A16, A24, and A32 address maps
- Establish the initial system hierarchy
- Configure the interrupt, trigger, and utility resources
- Initiate normal system operation

OBSERVATION F.1:

At power-on, the mainframes are isolated from each other, because the address windows and interrupt, trigger, and utility extension routing functions are disabled. The Resource Manager configures the system resources in the manner pre-determined by the system integrator, as described in Section E, "System Configuration."

F.1 LOGICAL ADDRESS WINDOW CONFIGURATION and DEVICE IDENTIFICATION

Given an adequate set of configuration guidelines, a Resource Manager can algorithmically configure the system's Logical Address Map. Typically, the configuration guidelines are a set of rules describing, for each type of bus in the system, the relationships between the various Logical Addresses. These include the parent bus extender's Logical Address and window range, the child bus extenders' Logical Addresses and window ranges, static configuration device addresses, and dynamic configuration device addresses.

A Resource Manager may configure a system's Logical Address map by recursively executing a Bus Configuration Procedure, beginning at the root bus. The Bus Configuration Procedure performs the following functions (using its configuration guidelines):

1. Find all SC devices and child-side extenders.
2. Configure any DC devices to appropriate addresses.
3. For each child-side extender:
   a. Open an appropriate downward Logical Address window.
   b. Execute the Bus Configuration Procedure for the child bus.
   c. Set the extender's downward window to match the child bus's Logical Address range.
4. Determine the bus's Logical Address range.
5. Return.

F.1.1 An Example

This section gives an example of how the Resource Manager identifies devices in a multiframe system and configures the Logical Address windows. The example system and configuration guidelines described in Section E.1.1, "System Configuration Example," are used here as well. The only prerequisite to proper Resource Manager operation is that the Logical Address of each SC device must be set by the system integrator before power-on, according to the example configuration guidelines.

The Resource Manager would perform the following steps:

Search for devices in Mainframe 1 in the range 00_{16} through FF_{16}.
(Find SC devices in the range 00_{16} through 77_{16}.)
(Find Mainframe Extenders **b** and **a** at Logical Addresses 76_{16} and 77_{16}.)
Move any DC devices to unused addresses in the range 1_{16} through 75_{16}.
Enable the Logical Address window of Mainframe Extender **b** with an outward range of 00_{16} through FF_{16}.

Search for devices on Interconnect Bus 2 in the range 78_{16} through FF_{16}.
(Find Mainframe Extender **h** at Logical Address 78_{16}.)
Enable the Logical Address window of Mainframe Extender **h** with an inward range of 00_{16} through FF_{16}.

Search for devices in Mainframe 6 in the range 79_{16} through FF_{16}.
(Find SC devices in the range 79_{16} through $7F_{16}$.)
Move any DC devices to unused addresses in the range 79_{16} through $7F_{16}$.

Because the Logical Address range in Mainframe 6 is 78_{16} through $7F_{16}$, set Extender **h**'s Logical Address window size to 8_{16}, and its base to 78_{16}.

Because the Logical Address range of Interconnect Bus 2 is 78_{16} through $7F_{16}$, set Extender **b**'s Logical Address window size to 8_{16}, and its base to 78_{16}.

Enable the Logical Address window of Mainframe Extender **a** with an outward range of 00_{16} through FF_{16}.

Search for devices on Interconnect Bus 1 in the range 80_{16} through FF_{16}.
(Find stand-alone devices A and B at Logical Addresses AC_{16} through AF_{16}.)
(Find Mainframe Extenders **g**, **d**, and **c** at Logical Addresses $B0_{16}$, $C0_{16}$, and $E0_{16}$.)
Enable the Logical Address window of Mainframe Extender **g** with an inward range of 00_{16} through FF_{16}.

Search for devices in Mainframe 5 in the range $B1_{16}$ through FF_{16}.
(Find SC devices in the range $B1_{16}$ through BF_{16}.).
Move any DC devices to unused addresses in the range $B1_{16}$ through BF_{16}.

Because the Logical Address range in Mainframe 5 is $B0_{16}$ through BF_{16}, set Extender **g**'s Logical Address window size to 10_{16}, and its base to $B0_{16}$.

Enable the Logical Address window of Mainframe Extender **d** with an inward range of 00_{16} through FF_{16}.

Search for devices in Mainframe 3 in the range $C1_{16}$ through FF_{16}.
(Find SC devices in the range $C1_{16}$ through CF_{16}.)
(Find Extender **e** at Logical Address $CF1_{16}$.)
Move any DC devices to unused addresses in the range $C1_{16}$ through CE_{16}.
Enable the Logical Address window of Mainframe Extender **e** with an outward range of $D0_{16}$ through FF_{16}.

Search for devices on Interconnect Bus 3 in the range $D0_{16}$ through FF_{16}.
(Find Mainframe Extender **f** at Logical Address $D0_{16}$.)
Enable the Logical Address window of Mainframe Extender **f** with an inward range of $D0_{16}$ through FF_{16}.

Search for devices in Mainframe 4 in the range $D1_{16}$ through FF_{16}.
(Find SC devices in the range $D1_{16}$ through DF_{16}.)
Move any DC devices to unused addresses in the range $D1_{16}$ through DF_{16}.

Because the Logical Address range in Mainframe 4 is $D0_{16}$ through DF_{16}, set Extender **f**'s Logical Address window size to 10_{16}, and its base to $D0_{16}$.

Because the Logical Address range of Interconnect Bus 3 is $D0_{16}$ through DF_{16}, set Extender **e**'s Logical Address window size to 10_{16}, and its base to $D0_{16}$.

Because the Logical Address range in Mainframe 3 is $C0_{16}$ through DF_{16}, set Extender **d**'s Logical Address window size to 20_{16}, and its base to $C0_{16}$.

Enable the Logical Address window of Mainframe Extender **c** with an inward range of $E0_{16}$ through FF_{16}.

Search for devices in Mainframe 2 in the range $E1_{16}$ through FF_{16}.
(Find SC devices in the range $E1_{16}$ through FE_{16}.)
Move any DC devices to unused addresses in the range $E1_{16}$ through FE_{16}.

Because the Logical Address range in Mainframe 2 is $E0_{16}$ through FF_{16}, set Extender **c**'s Logical Address window size to 20_{16}, and its base to $E0_{16}$.

Because the Logical Address range of Interconnect Bus 1 is 80_{16} through FF_{16}, set Extender **a**'s Logical Address window size to 80_{16}, and its base to 80_{16}.

F.2 A16, A24, and A32 SPACE ADDRESS MAP CONFIGURATION

Before configuring the A16, A24, and A32 address spaces, the Resource Manager first configures the system Logical Address map, as described in Section F.1. It then calculates the A16, A24, and A32 address range base and size for each extender as follows:

1. Starting at the bottom level of the tree and working up, add up the amount of memory needed by each mainframe and its descendants. At each level, round up the amount to the next power of 2 boundary allowed by the A16, A24, and A32 Window registers.

2. Starting at the root mainframe and working down each branch of the tree, assign memory beginning with the largest memory window requirements.

The Resource Manager then configures the windows using the calculated values.

OBSERVATION F.2:

VXI-1, Rev. 1.3 requires A24 and A32 bus masters to be capable of accessing the middle 3/4 of the respective address spaces.

RECOMMENDATION F.1:

When possible, the Resource Manager should avoid locating any memory resources within the A24 address ranges 0_{16} through $1FFFFF_{16}$ and $E00000_{16}$ through $FFFFFF_{16}$ and the A32 address ranges 0_{16} through $1FFFFFFF_{16}$ and $E0000000_{16}$ through $FFFFFFFF_{16}$.

OBSERVATION F.3:

A16 window configuration support is optional for Resource Managers.

F.3 SYSTEM SELF-TEST MANAGEMENT

System self-test management is handled as provided for in VXI-1, Rev. 1.3.

OBSERVATION F.4:

The Resource Manager can use the optional utility extensions SYSFAIL and SYSRESET to control and monitor the progress of system self-tests. If the Mainframe Extenders do not support utility extensions, the Resource Manager must wait at least five seconds after all mainframes are powered-on before testing the PASSED condition of each device.

F.4 SYSTEM HIERARCHY INITIALIZATION

The Resource Manager configures the initial system hierarchy as provided for in VXI-1, Rev. 1.3.

OBSERVATION F.5:

In some systems, the use of Logical Addresses to determine commander/servant hierarchies may conflict with the Logical Address window configuration requirements. In such cases, it is necessary to override the default hierarchy with a configuration table.

RECOMMENDATION F.2:

A multiframe Resource Manager should provide a means for the system integrator to supply commander/servant hierarchy configuration information.

F.5 INTERRUPT RESOURCE CONFIGURATION

The Resource Manager configures the interrupt resources as provided for in VXI-1, Rev. 1.3, with extensions for multiframe systems defined by the Interrupt Configuration register.

OBSERVATION F.6:

VXI-1, Rev. 1.3 requires a Resource Manager to provide a means for the system integrator to supply interrupt resource configuration information.

F.6 TTL and ECL TRIGGER and UTILITY RESOURCE CONFIGURATION

The Resource Manager configures these resources in an implementation-specific manner.

RECOMMENDATION F.3:

A multiframe Resource Manager should provide a means for the system integrator to supply TTL and ECL trigger resource configuration information. The Resource Manager should use such information to configure the optional trigger extension mechanisms defined by the TTL and ECL Trigger Configuration registers.

RECOMMENDATION F.4:

A multiframe Resource Manager should provide a means for the system integrator to supply utility bus configuration information. The Resource Manager would use such information to configure the optional utility bus extension mechanism defined by the Utility Configuration register.

F.7 INITIATION of NORMAL OPERATION

The Resource Manager initiates normal operation as provided for in VXI-1, Rev. 1.3.

APPENDIX I. REGISTER OVERVIEW:

This section is provided to improve the overview of the Mainframe Extender registers. Figure I.1 shows the Mainframe Extender register map.

Signature explanation:

[ ]: Optional register
( ): Required for Slot 0 Mainframe Extender

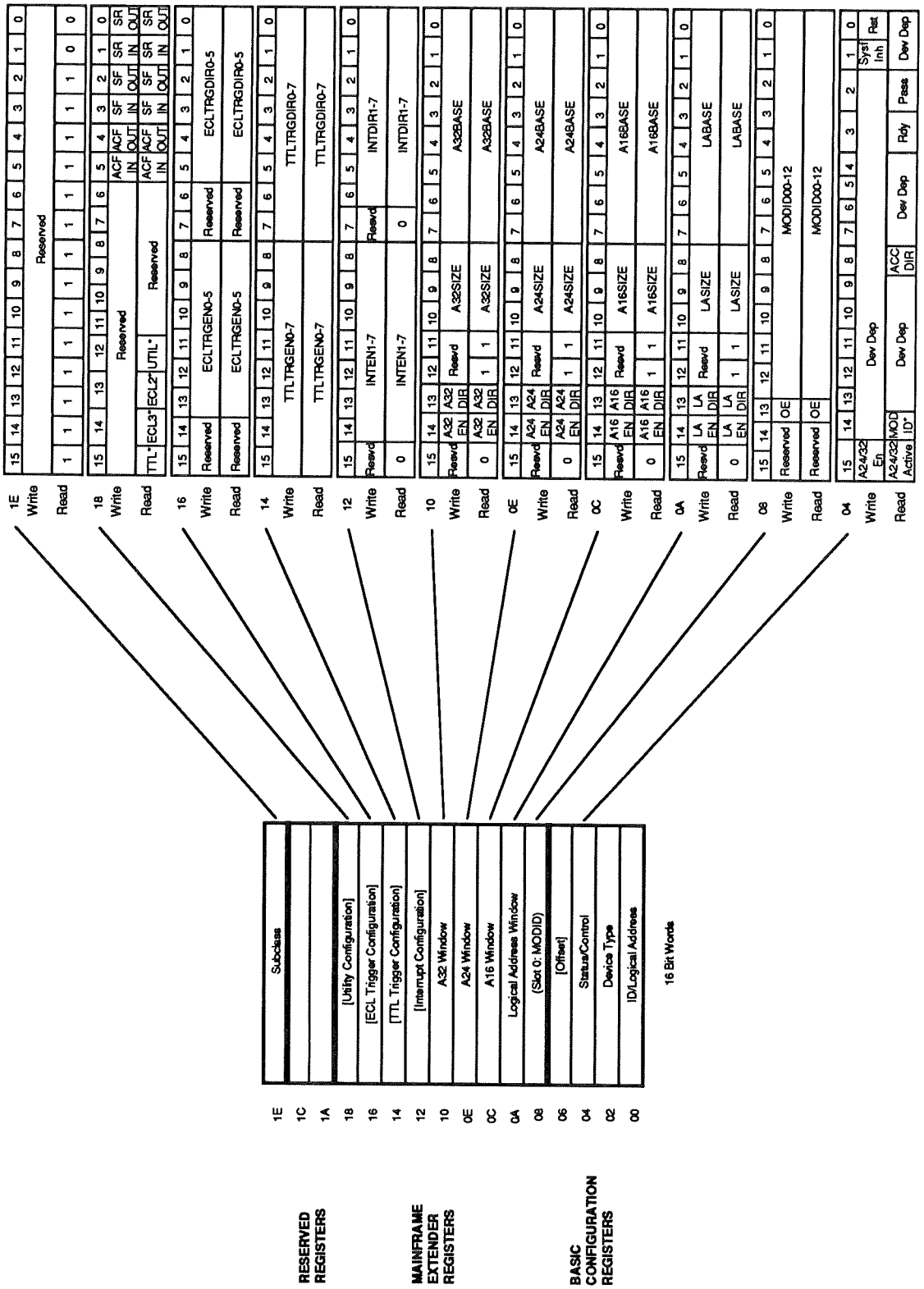
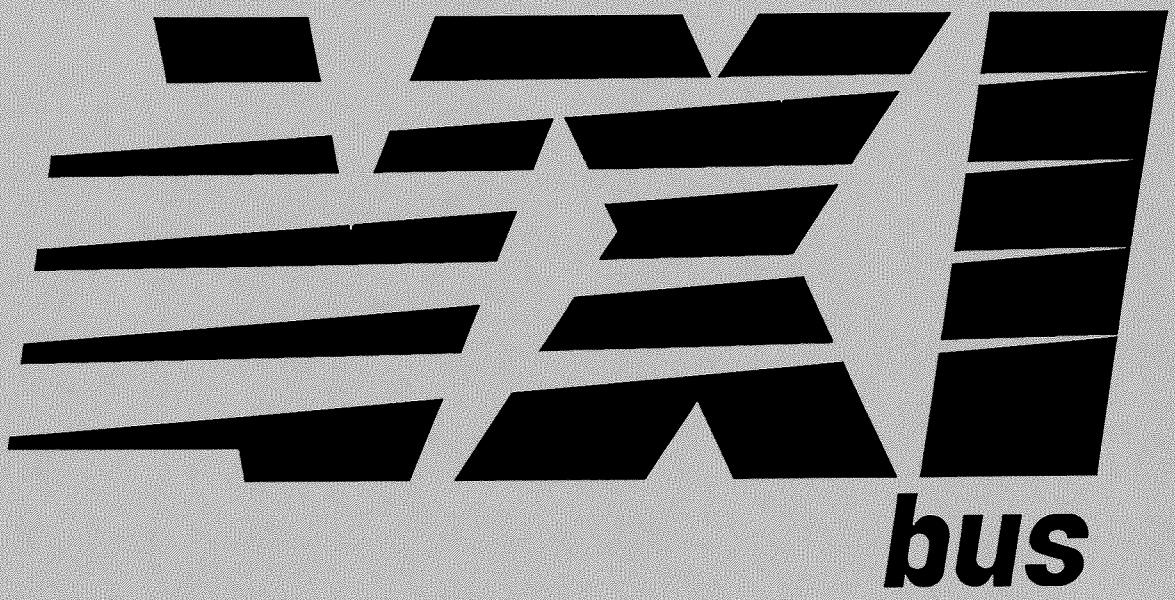


Figure I.1. Register Overview for Mainframe Extender.

VMEbus Extensions for Instrumentation:



Shared Memory Data Formats

Specification

VXI-7

REVISION: 1.0

March 1, 1992

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extensions for Instrumentation: Shared Memory Data Formats Specification, VXI-7 Revision 1.0 is authored by the VXIbus Consortium, Inc. and its sponsor members:

A/S Bruel & Kjaer
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments, Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce and distribute this document by any means for any purpose.

CONTENTS

A. INTRODUCTION 1
B. DEFINITIONS 2
C. GENERAL PRINCIPLES 3
D. DATA ACCESS CONVENTIONS 4
D.1 ADDRESSING 4
D.2 DATA ALIGNMENT 4
D.2.1 Block Boundaries 4
D.2.2 Packed Blocks 5
D.2.3 Padded Blocks 5
D.3 BIT AND BYTE ORDERING 5
E. DATA ELEMENT FORMATS 6
E.1 NUMERIC ELEMENTS 6
E.2 CHARACTER ELEMENTS 6
E.3 ARBITRARY ELEMENTS 6
E.4 OTHER ELEMENTS 7
F. DATA STRUCTURES 8
F.1 ARRAYS 8
F.2 CHARACTER STRINGS 8

VMEbus Extensions for Instrumentation:

Shared Memory Data Formats

A. INTRODUCTION

The architecture of the VXIbus allows devices to communicate via global RAM. This is accomplished by one device writing data to memory, followed by another device reading the data. In order for this mechanism to work properly, the following conditions are necessary:

1. Both devices must use the same format for each data element passed.
2. Both devices must agree on the overall data structure of the passed information.
3. Both devices must agree on the memory addresses used for communication.
4. Both devices must agree on the ordering of the data within the defined memory addresses.
5. The devices must synchronize their accesses to the memory.

Data structures, synchronization, and address bounds are all determined by protocols. Data formats and data ordering can be defined independently from any protocols. This document is an addendum to the VXIbus system specification which defines common data formats for transferring information between VXIbus devices via global RAM. Included are discussions of data access conventions, definition of low level data element formats, and definition of common terminology for combining data elements into more-complex data structures.

This standard is compatible with:

- VMEbus Extensions for Instrumentation System Specification, Revision 1.3, July 14, 1989, hereafter referred to as VXI-1 Rev. 1.3.
- ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.
- ANSI X.3.4-1986, American National Standard Code for Information Interchange Coded Character Set - 7-bit.
- ANSI X.3.41-1974, American National Standard Code Extension Techniques for Use with the 7-bit Coded Character Set of American National Standard Code for Information Interchange.

B. DEFINITIONS

The following terms are used extensively throughout this specification. While their general definitions are well understood, they have very specific meanings within this context. They provide an unambiguous vocabulary for use in documenting the structure of shared memory data.

Data Element: A data element is an indivisible unit, such as a single numeric quantity or an alphanumeric character.

Data Structure: A data structure is any orderly grouping of data elements, data arrays, or data structures.

Data Array: A data array is an ordered grouping of data structures or data elements, termed *components*, all of which have the same internal structure or format.

Component: A component of an array is one of the ordered data elements or data structures comprising that array.

Block: A block is a data structure containing a complete "message" to be transferred from one device to another

Pad: A pad is a sequence of one or more bytes of unspecified format used to force alignment of data at optimal boundaries. No information is encoded within pads.

Most Significant Part: The most significant part of a data element or structure is that portion which represents the most important part of the information represented. For numeric quantities, this is the bit (or byte) representing the largest quantity. For ordered sequences of data, this is the data element which is logically "first", in order of generation and/or consumption. In some cases the choice is entirely arbitrary.

Least Significant Part: The least significant part of a data element or structure is that portion which represents the least important part of the information represented. See the definition of *Most Significant Part*.

C. GENERAL PRINCIPLES

This specification is based as much as possible on common conventions among users of the VXIbus and the VMEbus, as well as IEEE/ANSI standards. In general, the following principles apply:

1. Numeric quantities are expressed in IEEE 754-1985 defined formats.
2. Data is ordered starting at the lowest memory address in use.
3. Data is always transferred with its least significant bit carried by the D0 data line.
4. The least significant byte of a data element is always in the highest-numbered memory address.
5. Data elements and structures should start on optimal address boundaries.

D. DATA ACCESS CONVENTIONS

D.1 ADDRESSING

Each memory location has a unique address within its address space (A16, A24, or A32) determined by the VMEbus address and control lines. The 2 least significant bits of the address are determined by the lines A01, DS0*, DS1*, and LWORD*, as defined by the VMEbus specification. The remaining bits correspond to the address lines A02-A63, with A02 corresponding to the 3rd least significant bit, and the remaining bits arranged sequentially. A high level on an address line represents a value of one for the corresponding bit of the address. A low level represents a value of zero.

OBSERVATION D.1.1:
The preceding description is simply a statement of the conventions implied by the VMEbus standard.

RULE D.1.1:
All blocks of data **SHALL** be written such that the first data element of the block is at the lowest address within that block, and succeeding data elements are at successively higher addresses.

OBSERVATION D.1.2:
The preceding rule provides that start of a block of data (whose format is not specified here) is at its lowest address, and its end is at its highest address.

D.2 DATA ALIGNMENT

The alignment of data elements and blocks is a major factor in the efficiency of data transfer and processing. In applications requiring the highest data processing speeds, each data element should be aligned such that it may be individually transferred in the fewest possible bus cycles. For example, a long word (32 bits) should be aligned for access by a single longword bus cycle (starting at an address divisible by 4). This property is embodied in *padded* blocks, in which the data elements are aligned along their most natural boundaries (with pads included where necessary).

When data transfer rates must be maximized, the data block and its elements should be aligned such that the whole block may be copied in the fewest number of bus cycles. In this case, the block should begin on a longword (divisible by 4) boundary and contain the smallest possible multiple of 4 bytes. This application is best served by a *packed* block, in which the data elements are stored in contiguous locations.

D.2.1 Block Boundaries

RULE D.2.1:
Blocks **SHALL** start on addresses that are divisible by four (4), **AND** be multiples of four (4) bytes in length.

OBSERVATION D.2.1:
With the above alignment both D16 and D32 VMEbus transfers can be supported.

OBSERVATION D.2.2:
The VXIbus specification requires support of D16 data transfers. All other modes are optional.

D.2.2 Packed Blocks

Within a packed block, data elements and structures are stored in contiguous memory locations. No pads are inserted between any elements or structures.

OBSERVATION D.2.3:
Within a packed block, elements and structures will not be aligned along their natural boundaries.

OBSERVATION D.2.4:
Within a packed block, all data structures are also packed.

OBSERVATION D.2.5:
A packed block will include enough pad bytes (0-3) at its end to force its size to a multiple of 4 bytes.

D.2.3 Padded Blocks

Within a padded block, all elements and structures are aligned along their optimal boundaries. Pads are included to force this alignment.

RULE D.2.2:
Within a padded block, any data element or structure consisting of exactly two (2) bytes **SHALL** start on an address that is divisible by two (2).

RULE D.2.3:
Within a padded block, any data element or structure having more than two (2) bytes **SHALL** start on an address that is divisible by four (4), **AND** be padded such that the next element starts on an address that is divisible by four (4).

OBSERVATION D.2.6:
Within a padded block, all data structures are also padded.

RULE D.2.4:
A padded data block **SHALL NOT** include any unnecessary pads.

D.3 BIT AND BYTE ORDERING

Each VMEbus read or write uses 8, 16, or 32 of the VMEbus data lines (D00-D31) to transfer the data. Within each transfer, the least significant data bit is transferred on the lowest numbered data line, with the remaining bits arranged sequentially on the corresponding higher numbered data lines.

Many data elements require more than one byte of memory. They always occupy successive memory locations, with the most significant byte at the lowest address, and the remaining bytes arranged in order of decreasing significance.

E. DATA ELEMENT FORMATS

RULE E.0.1:

Each data block **SHALL** contain at least one element conforming to one of the common **Numeric** or **Character** element formats defined in the following sections.

E.1 NUMERIC ELEMENTS

RECOMMENDATION E.1.1:

Each data element representing a numeric quantity should have one of the following common numeric formats:

8-bit signed integer: A number in the range of -128 to 127 which is stored as an 8 bit 2's complement value.

16-bit signed integer: A number in the range of -32768 to 32767 which is stored as a 16 bit 2's complement value.

32-bit signed integer: A number in the range of -2147483648 to 2147483647 which is stored as a 32 bit 2's complement value.

8-bit unsigned integer: A number in the range of 0 to 255 which is stored as an 8 bit binary value.

16-bit unsigned integer: A number in the range of 0 to 65535 which is stored as a 16 bit binary value.

32-bit unsigned integer: A number in the range of 0 to 4294967295 which is stored as a 32 bit binary value.

32-bit Real: IEEE 754-1985 32 bit single format real value.

64-bit Real: IEEE 754-1985 64 bit double format real value.

E.2 CHARACTER ELEMENTS

RECOMMENDATION E.2.1:

Each data element representing a character should have one of the following common character formats:

7-bit character: An ASCII encoded character as defined by ANSI X3.4-1986. This is stored as a single byte wherein the most significant bit is set to 0.

8-bit character: An ASCII encoded character as defined by ANSI X3.41-1974.

E.3 ARBITRARY ELEMENTS

RECOMMENDATION E.3.1:

Each data element representing a data type not suited to the common numeric or character formats should have one of the following arbitrary formats:

8-bit arbitrary: A single byte of data. Its format is application dependent.

16-bit arbitrary: 2 bytes of data. Its format is application dependent.

32-bit arbitrary: 4 bytes of data. Its format is application dependent.

E.4 OTHER ELEMENTS

PERMISSION E.4.1:

Other data formats may be used as required by the applications. These formats are application dependent. They are to be stored as sequences of bytes, with the first byte at the lowest address and the following bytes at successively higher addresses.

F. DATA STRUCTURES

A data structure is an ordered set of dissimilar data elements and/or data structures. Each data structure has a defined format, which may be application specific. A structure's first element starts at the start address (lowest address) of that data structure. The following elements are located at ascending addresses. If the structure is packed, all of its elements occupy contiguous memory locations. If the structure is padded, each element is aligned along its natural boundary.

Two of the most common data structures are *arrays* and *character strings*. They are described in the following paragraphs.

F.1 ARRAYS

An array is a sequential grouping of like components. It may have one or more dimensions.

Single-Dimension Array: A single-dimension array, A(i), has I components designated a(0), a(1), ..., a(i-2), a(i-1) arranged sequentially in ascending memory.

Two-Dimension Array: A two-dimension array, A(i, j) is an array of i arrays of j components each.

N-dimension Array: The concept of a two-dimension array may be extended to the general case of the n-dimension array. An n-dimension array A($i,j,...$) is an array of i $(n-1)$-dimension arrays.

F.2 CHARACTER STRINGS

A character string is a data structure including an ordered sequence of characters arranged sequentially in ascending memory locations. It may have one of several formats.

Fixed-length string: A fixed-length string is an array of n characters. The string length, n, is fixed by the application.

Counted string: A counted string is a character sequence of arbitrary length. The string length is encoded in a 32-bit unsigned integer which immediately precedes the string itself.

NULL Terminated string: A NULL terminated string is a character sequence of arbitrary length, followed by the ASCII encoded NULL character (binary value 0).

OBSERVATION F.2.1:

It is possible to embed strings of different types within a given character string. For example, a fixed-length string may consist of a NULL terminated string followed by pad bytes, or a NULL terminated string may consist of a series of CR-LF (Carriage return - Line feed) terminated strings.

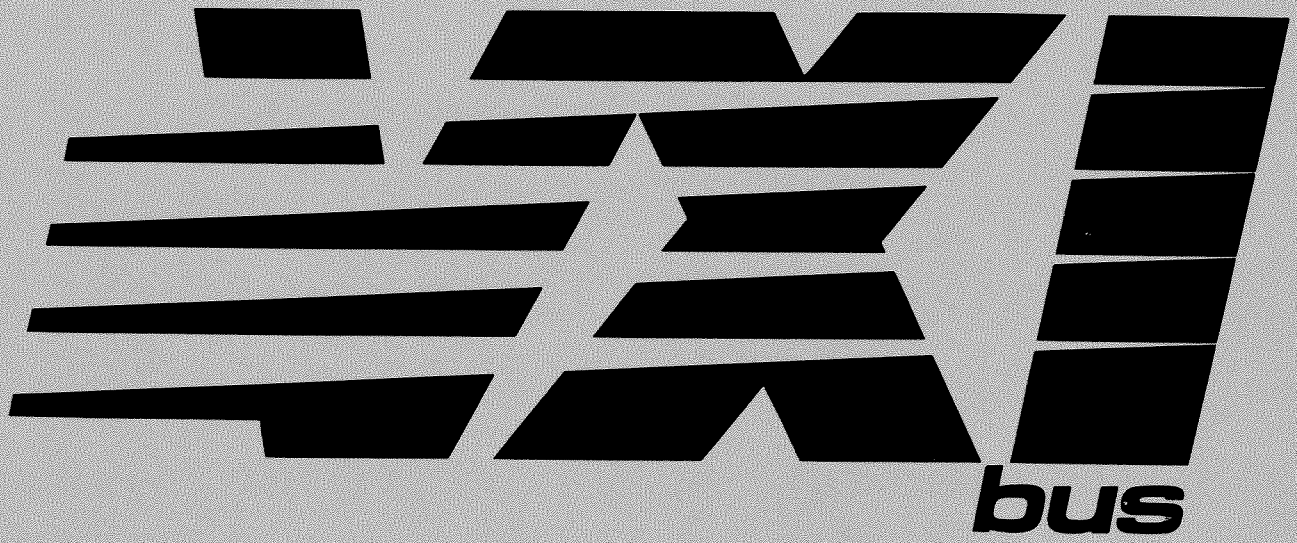
OBSERVATION F.2.2:

The use of counted or NULL terminated strings as major elements of a block will result in the block having a variable length. Such a block cannot be copied without parsing its strings, which can result in reduced data transfer rates.

RECOMMENDATION F.2.1:

When possible, counted and NULL terminated strings should be embedded within fixed-length strings.

VMEbus Extension for Instrumentation



Shared Memory Communication Protocol

Specification

VXI-9

REVISION 1.0

May 22, 1992

NOTICE

The information contained in this document is subject to change without notice.

The VXIbus Consortium, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The VXIbus Consortium, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

VMEbus Extension for Instrumentation, Shared Memory Communication Protocol Specification VXI-9, Revision 1.0, is authored by the VXIbus Consortium, Inc. and its sponsor members:

Bruel & Kjaer Test Systems A/S
Colorado Data Systems, Inc.
John Fluke Mfg. Co., Inc.
GenRad, Inc.
Hewlett-Packard Co.
Keithley Instruments, Inc.
National Instruments Corp.
Racal-Dana Instruments, Inc.
Tektronix, Inc.
Wavetek, Inc.

This document is in the public domain. Permission is granted to reproduce this document by any means for any purpose.

CONTENTS

A. INTRODUCTION 1
A.1 MEMORY ALIGNMENT 1
B. DEFINITIONS 2
C. PROTOCOL SERVICES 3
C.1 OVERVIEW 3
C.2 SPECIFICATION 3
C.2.1 Establish Channel 4
C.2.2 Establish APID 5
C.2.3 Delete APID 5
C.2.4 Verify APID 5
C.2.5 Acquire Buffers 6
C.2.6 Send Buffers 7
C.2.7 Receive Buffers 7
C.2.8 Return Buffers 8
C.2.9 Channel Test 8
C.2.10 Reclaim Buffers 9
D. SHARED MEMORY DEVICES 10
D.1 GENERAL REQUIREMENTS 10
D.2 SERVER REQUIREMENTS 11
D.3 CLIENT REQUIREMENTS 14
E. CHANNEL ESTABLISHMENT 15
E.1 CHANNEL SETUP DATA AREA 15
E.2 CHANNEL SETUP DATA STRUCTURE 16
E.3 CONNECTION PROCEDURE 17
E.4 EXCEPTION HANDLING 19
F. CHANNEL OPERATION 22
F.1 OPERATION OVERVIEW 22
F.1.1 Data Structures Used 22
F.1.2 Transferring Data 22
F.1.3 Using Buffers 22
F.1.4 Using Packets 22
F.1.5 Using Queues 23
F.2 BUFFERS 23
F.2.1 Buffer Header 23
F.2.2 Buffer Data Area 25
F.2.3 Buffer Lists 25
F.2.4 Lists of Buffer Lists 26
F.3 PACKETS 26
F.4 QUEUES 27
F.5 CHANNEL OPERATION 28
F.5.1 Packet Handling 28
F.5.2 Packet Error Handling 30
F.5.2.1 UNRECOGNIZED COMMANDS 30
F.5.2.2 INVALID MEMORY ADDRESSES 30
F.5.2.3 UNRECOGNIZED ACCESS POINT IDENTIFIER 31
F.5.3 Buffer Management 31
F.5.3.1 ACQUIRING BUFFERS 31
F.5.3.2 SENDING DATA 34
F.5.3.3 RETURNING BUFFERS 34

F.5.4 Example: Client Writes to Server 35
F.5.5 Example: Server Writes to Client 35
F.6 CHANNEL UTILITIES 35
F.6.1 Channel Test 35
F.6.2 Access Point Identifier Resolution 36
F.6.3 Address Range Resolution 36
F.7 CHANNEL ABORT 37
F.8 CHANNEL TERMINATION 38
G. PACKET COMMANDS 39
H. PROTOCOL EVENTS 43
APPENDIX I. PROTOCOL STATE TABLES 46
APPENDIX II. REVISION 1.2 COMPATIBILITY 50

LIST OF TABLES

TABLE E.1. Connection Protocol 18
TABLE I.1. Client State Table 47
TABLE I.2. Server State Table 48
TABLE I.3. Server+Client State Table 49

VMEbus Extension for Instrumentation
Shared Memory Communication Protocol

A. INTRODUCTION

This document is an addendum to the VXIbus System Specification. It describes the rules for certain devices which communicate using shared VMEbus memory. Shared memory can be used for transferring large amounts of data efficiently. An example is the transfer of an acquired waveform between a data acquisition device and an embedded controller. Shared memory can also be used to communicate outside the commander/servant relationship.

The SHARED MEMORY Communication Protocol is intended to be the lowest layer in a layered communication architecture. It defines an unreliable datagram service for communication between devices in a VXIbus environment. Definition of higher layers, such as Distributed Network Support, may be added in the future.

This specification is compatible with:

- THE VMEbus Specification Revision C.1, Micrology pbt.
- VMEbus Extension for Instrumentation System Specification, Revision 1.3, July 14, 1989, hereafter referred to as VXI-1, Rev. 1.3.

A.1 MEMORY ALIGNMENT

RULE A.1.1:
All longword (4 byte) memory structures (such as addresses or pointers) defined in the SHARED MEMORY Communication Protocol **SHALL** have the following alignment.

02_{16}	Least Significant Word
00_{16}	Most Significant Word

This alignment assures that the most significant bit of the longword will be on the D31 data line in a D32 data transfer.

All addresses defined in this document are VMEbus addresses. Also, unless otherwise specified, all values are decimal.

B. DEFINITIONS

Access Point Identifier: An identifier which uniquely specifies a Higher Layer Entity. It is used in the interface between the SHARED MEMORY Communication Protocol layer and the Higher Layer Entity.

Buffer: The data structure used by the SHARED MEMORY Communication Protocol to transfer data between two SHARED MEMORY devices. A buffer consists of a Buffer Header and an associated Buffer Data Area. The Header contains control information and the Data Area contains the actual data being transferred.

Buffer List: A linked list of SHARED MEMORY buffers. Lists are linked via the Next Buffer Address field in the Buffer Header.

Channel: A SHARED MEMORY channel is a full-duplex data path between two VXIbus devices. The SHARED MEMORY Communication Protocol supports up to two channels between a SHARED MEMORY Server device and a SHARED MEMORY Client device. These are the Synchronous channel for normal data transfer and the Asynchronous channel for expedited data transfer.

Channel Setup Data Structure: The data structure used to set up a channel between two SHARED MEMORY devices. The SHARED MEMORY Server Device uses the Channel Setup Data Structure to pass setup information to the Client Device.

Client Device: The SHARED MEMORY device which participates in a channel but does not control the shared memory used in the channel.

Command: a 16-bit value containing SHARED MEMORY Communication Protocol control information being transferred between two SHARED MEMORY devices.

Command Packet: The data structure used for the transfer of SHARED MEMORY Communication Protocol commands between two SHARED MEMORY devices.

Higher Layer Entity: A function within a VXIbus device which utilizes the services of the SHARED MEMORY Communication Protocol to communicate with a corresponding entity on another device.

List of Buffer Lists: A linked list of SHARED MEMORY buffer lists. Lists of buffer lists are linked via the Next List Address field in the Buffer Header.

Queue: A circular buffer of command packets used to transfer commands between two SHARED MEMORY devices. Two queues are associated with each SHARED MEMORY channel: an input queue and output queue. The queues are owned by the Server device.

Server Device: The SHARED MEMORY device which has control of the shared memory used in a channel. It is responsible for configuring the shared memory required for a particular channel, during setup.

C. PROTOCOL SERVICES

Each shared memory implementation must provide a minimum set of services in order to guarantee support of common protocol hierarchies. The interface descriptions are used as a means of specifying the functionality of the service, and are not intended to constrain the nature of the interface itself. In particular, the interface descriptions assume an implementation that provides the user direct access to the shared memory buffers. Some implementations may have interfaces which protect the buffers from direct user access.

C.1 OVERVIEW

RULE C.1.1:
Each shared memory implementation **SHALL** provide the following services to its users:

- ESTABLISH CHANNEL — This service establishes a SHARED MEMORY Communication Protocol channel to a specified device.
- ESTABLISH APID — This service identifies the user as the recipient of all incoming buffers directed to a specified Access Point Identifier (APID).
- DELETE APID — This service allows a user to mark a previously established APID as unused.
- VERIFY APID — This service determines whether or not the specified APID is currently available at a specified device.
- ACQUIRE BUFFERS — This service provides the user access to one or more lists of empty buffers to be filled and sent to a specified device.
- SEND BUFFERS — This service sends a list of buffers to a specified destination (Device + APID).
- RECEIVE BUFFERS — This service provides the user access to a list of incoming data buffers directed to the specified APID.
- RETURN BUFFERS — This service allows the user to relinquish access to a list of empty buffers.
- CHANNEL TEST — This service allows a user to initiate a channel test sequence.
- RECLAIM BUFFERS — This "service" provides a mechanism for the shared memory implementation to asynchronously require the user to relinquish access to all data buffers used for communication with the specified device.

C.2 SPECIFICATION

The following sections describe each service, along with its user inputs and outputs. Each input or output is listed in the following format:

name [range of values]

These service specifications are to be considered as general descriptions of the capabilities required of the SHARED MEMORY Communication Protocol. Particular implementations are not required to partition the services as described here. For a more complete understanding of this specification, refer to the details of the SHARED MEMORY Communication Protocol, in subsequent sections.

C.2.1 Establish Channel

INPUTS:	Channel_type	[Asynchronous, Synchronous]
	Address_Space	[A24, A32, Any]
	Remote_Ladd	[0 .. 255]
	Role	[Client, Server, Any]
OUTPUTS:	Status	[OK, Exists, Error]
	Address_Space	[A24, A32]
	Data_Width	[D08, D16, D32, D32B, D64]
	Role	[Client, Server]
	Error_Code	[(implementation dependent)]

DESCRIPTION: This service attempts to establish a shared memory channel of the type Channel_type, in the address space Address_space, with the device at logical address Remote_ladd. If the procedure is successful, the service will return OK for the Status. If a channel with the device at remote_ladd already exists, the Status Exists will be returned. In either case the returned values Address_Space, Data_Width and Role indicate the actual channel configuration. If such a channel cannot be established, the service returns Error for the status. In this case, the Error_code indicates the reason for the failure. The known causes of failure include the following:

There is no device at Remote_ladd.
A channel can't be established in Address_space.
A Channel_type channel can't be established.
Insufficient CSDS resources available.
Insufficient channel resources available.
Device at Remote_ladd does not support SHARED MEMORY Communication Protocol.
Device at Remote_ladd is not in Ready state.
Device cannot be a client.
Device cannot be a server.

PARAMETERS: Channel_type - Indicates whether the channel should be a Synchronous or an Asynchronous channel.
Address_Space - Indicates whether the channel is established using VMEbus A24 or A32 memory.
Remote_ladd - The VXIbus logical address of the device at the remote end of the SHARED MEMORY channel.
Role - Indicates whether the device takes the role of client or server for this channel.

C.2.2 Establish APID

INPUTS: APID_number [0 .. $FFFFFFFF_{16}$]

OUTPUTS: Status [OK, Error]
Error_Code [(implementation dependent)]

DESCRIPTION: This service enables the shared memory implementation to receive data buffers addressed to APID_number, thus establishing a valid access point for the SHARED MEMORY Communication Protocol. If the access point is successfully established, OK is returned as the Status. Otherwise, Error is returned. The known causes of error include the following:

The APID has already been established.

PARAMETERS: APID_number - The Access Point Identifier number to be established.

C.2.3 Delete APID

INPUTS: APID_number [0 .. $FFFFFFFF_{16}$]

OUTPUTS: Status [OK, Error]
Error_Code [(implementation dependent)]

DESCRIPTION: This service disables the shared memory implementation from receiving any data buffers addressed to APID_number, thus deleting a valid access point for the device's SHARED MEMORY Communication Protocol. If the access point is successfully deleted, OK is returned as the Status. Otherwise, Error is returned. The known causes of error include the following:

The APID is not currently established.

PARAMETERS: APID_number - The Access Point Identifier number to be deleted.

C.2.4 Verify APID

INPUTS: APID_number [0 .. $FFFFFFFF_{16}$]
Remote_ladd [0 .. 255]

OUTPUTS: Status [Available, Unavailable, Error]
Error_Code [(implementation dependent)]

DESCRIPTION: This service uses the Access Point Resolution protocol to determine if APID_number is currently available on the device at logical address Remote_ladd. If the Access point is available, Available is returned as the Status, else Unavailable is returned. If the Access Point Resolution protocol cannot be executed, Error is returned. The known causes of failure include:

There is no channel active with the device at Remote_ladd.

PARAMETERS: APID_number - The Access Point Identifier number to be checked.
Remote_ladd - The VXIbus logical address of the device to be for the specified APID.

C.2.5 Acquire Buffers

INPUTS:

Dest_ladd	[0 .. 255]
Num_of_lists	[1 .. $FFFF_{16}$]
Bytes_per_list	[1 .. $FFFFFFFF_{16}$]

OUTPUTS:

List_address	
Status_flags	[OK, Partial, Request_too_big, Error]
Error_Code	[(implementation dependent)]

DESCRIPTION: This service provides the user with access to a list of empty buffer lists to be used to send data to the device at logical address Dest_ladd. The service attempts to provide Num_of_lists lists of buffers, where each list has at least Bytes_per_list bytes of capacity. If the requested quantity of free buffers is not available, the request will be partially filled. The address of the first buffer in the list of buffer lists is returned as List_address. If the request is completely satisfied, OK is returned as the Status. If the request is only partially filled, either Partial or Request_too_big is returned. Request_too_big is returned only if the requested quantity of free buffers exceeds the total buffer capacity available to the channel. If no buffers are returned, the Error flag is set. The known error causes include the following:

There is no channel active with the device at Dest_ladd.
There are no free buffers currently available.

PARAMETERS: Dest_ladd - The VXIbus logical address of a VXIbus device. The buffers will be used to send data to this device.
Num_of_lists - The number of buffer lists being requested. Each of the lists has at least the number of bytes specified by the parameter Bytes_per_list.
Bytes_per_list - This is the minimum capacity (in bytes) of each of the lists being requested.
List_address - This is the pointer to the first buffer in the list of buffer lists.

C.2.6 Send Buffers

INPUTS:	APID_number	[0 .. $FFFFFFFF_{16}$]
	Dest_ladd	[0 .. 255]
	Channel_Type	[Asynchronous, Synchronous]
	List_Address	
OUTPUTS:	Status_flags	[OK, Error]
	Error_Code	[(implementation dependent)]

DESCRIPTION: This service sends a buffer list to a remote device. The buffer list is sent as a single unit, without fragmentation or concatenation to any other buffer list. It is implementation dependent whether the destination logical address (dest_ladd) and access point identifier (APID_number) are passed as parameters or within the buffer header(s). The first buffer in the list is at address List_address. The list is transmitted via the Channel_type channel to device Dest_ladd. Once the buffers have been sent (inserted in the message queue) the service will return OK as the Status. If the buffer list cannot be sent, Error is returned as the Status. The known error causes include the following:

There is no channel active with the device at Dest_ladd.
The buffer list contains invalid address pointers.
The buffer list includes a buffer which cannot be sent to the device at Dest_ladd.

PARAMETERS: Channel_type - Indicates whether the buffers are to be sent via the Synchronous or Asynchronous channel.
APID_number - The Access Point Identifier number to be to be sent in the buffer headers. This identifies the remote access point.
Dest_ladd - The VXIbus logical address of a VXIbus device. The buffers will sent to this device.
List_address - This is the pointer to the first buffer in the buffer list.

C.2.7 Receive Buffers

INPUTS:	APID_number	[0 .. $FFFFFFFF_{16}$]
	Source_ladd	[0 .. 255, Any]
OUTPUTS:	Source_ladd	[0 .. 255]
	Channel_Type	[Asynchronous, Synchronous]
	List_Address	
	Status_flags	[OK, Error]
	Error_Code	[(implementation dependent)]

DESCRIPTION: This service provides the user access to a data buffer list which has been received from some other device. The buffer list is provided as a single unit, without fragmentation or concatenation to any other buffer list. The buffer lists for each channel type are provided in order of receipt. Buffers from Asynchronous channels always take precedence over buffers from Synchronous channels. When a buffer list becomes available, the service will return the address of the first buffer (List_address), the logical address of the sending device, and the type of channel used to send the buffer list (Channel_type), and OK as the Status. It is implementation dependent whether the source logical address (Source_ladd) is passed as a parameter or within the buffer header(s). The known error causes include the following:

The Access Point APID_number is not active.

PARAMETERS: Channel_type - Indicates whether the buffers were received via the Synchronous or Asynchronous channel.
APID_number - The Access Point Identifier number in the buffer headers of the buffer list requested.
Source_ladd - The VXIbus logical address of the device sending the buffer list.
List_address - This is the pointer to the first buffer in the buffer list.

C.2.8 Return Buffers

INPUTS: List_Address
Address_Space [A24, A32, Local]

OUTPUTS: Status [OK, Error]
Error_Code [(implementation dependent)]

DESCRIPTION: This service allows the user to relinquish access to a list of empty buffer lists. The service returns the buffers to the appropriate free buffer pool and returns OK as the Status.

PARAMETERS: List_address - This is the pointer to the first buffer in the buffer list.
Address_Space - Indicates whether the buffers returned are from local memory, remote memory in the A24 address space or remote memory in the A32 address space.

C.2.9 Channel Test

INPUTS: Dest_ladd [0 .. 255]
Channel_Type [Asynchronous, Synchronous]

OUTPUTS: Status [OK, Error]
Error_Code [(implementation dependent)]

DESCRIPTION: This service allows the user to execute the Channel Test protocol. A TEST command packet is sent to the device at logical address Dest_ladd, using the Channel_type channel. Upon the receipt of the matching TEST_ACK packet, the service returns OK as the Status. If the Channel Test protocol cannot be executed, Error is returned. The known causes of error include:

There is no active channel with the device at Dest_ladd.

PARAMETERS: Dest_ladd - The VXIbus logical address of a VXIbus device. The channel to be tested connects to this device.
Channel_type - Indicates whether the channel to be tested is a Synchronous or Asynchronous channel.

C.2.10 Reclaim Buffers

OUTPUTS:	Remote_ladd	[0 .. 255]
	Action	[Return, Abandon]
INPUTS:	Status	[OK , No_buffers]
	List_address	

DESCRIPTION: This service asynchronously requests that the user either return (Action = Return) or stop accessing (Action = Abandon) all buffers used to communicate with the device at logical address Remote_ladd. If the request is to return the buffers, the user must return the buffers. If the request is to abandon the buffers, the user simply ceases access of the buffers. After complying with the request, the user returns OK as the status. If the user has no buffers, No_buffers is returned.

PARAMETERS: Remote_ladd - The VXIbus logical address of a device. The reclaimed buffers are used to communicate with this device.
Action - Indicates whether the buffers are to be returned or abandoned.
List_address - This is the pointer to the first buffer in the returned buffer list.

D. SHARED MEMORY DEVICES

SHARED MEMORY devices support the VXIbus SHARED MEMORY Communication Protocol.

D.1 GENERAL REQUIREMENTS

SHARED MEMORY devices communicate through SHARED MEMORY channels. A channel is the shared memory and protocol used by two SHARED MEMORY devices to communicate.

RULE D.1.1:
A SHARED MEMORY device **SHALL** be a message based device.

PERMISSION D.1.1:
The SHARED MEMORY Communication Protocol **MAY** be supported by VXIbus devices which are not message based. Establishing SHARED MEMORY channels with such devices requires device dependent knowledge of the SHARED MEMORY Communication Protocol implementation on those devices.

RULE D.1.2:
A SHARED MEMORY device **SHALL** know its own logical address.

RULE D.1.3:
A SHARED MEMORY device **SHALL** be capable of sending events to and receiving events from other SHARED MEMORY devices.

OBSERVATION D.1.1:
SHARED MEMORY events may be either VXIbus signals or VMEbus interrupts.

RECOMMENDATION D.1.1:
Because of the limited nature of interrupts, it is recommended that all SHARED MEMORY devices implement a Signal Register. The use of VXIbus signals will reduce the load on the limited VXIbus interrupt resources, and provide a simpler, symmetric handshaking mechanism.

RULE D.1.4:
A SHARED MEMORY channel **SHALL NOT** be shared by more than two SHARED MEMORY devices.

RULE D.1.5:
One of the SHARED MEMORY devices sharing a channel **SHALL** implement the SHARED MEMORY Server functions. The other SHARED MEMORY device **SHALL** implement the SHARED MEMORY Client functions.

PERMISSION D.1.2:
Any SHARED MEMORY device **MAY** maintain channels to any number of SHARED MEMORY devices at any time.

RULE D.1.6:
Two SHARED MEMORY devices **SHALL NOT** share more than one Synchronous channel and one Asynchronous channel.

RULE D.1.7:
A SHARED MEMORY channel **SHALL** be established in an area of memory accessible to both SHARED MEMORY devices and controlled by the SHARED MEMORY device implementing the SHARED MEMORY Server function.

OBSERVATION D.1.2:
This requirement can be met in a number of ways:

- Both devices are A24 masters and the memory resides on a third device and is A24 accessible. This memory must be exclusively controlled by the SHARED MEMORY Server device (either through commander/servant assignment or some other allocation scheme).

- Both devices are A32 masters and the memory resides on a third device and is A32 accessible. This memory must be exclusively controlled by the SHARED MEMORY Server device (either through commander/servant assignment or some other allocation scheme).

- One of the devices is an A24 master, and the memory resides on the other device (the SHARED MEMORY Server device) and is A24 and internally (dual port) accessible.

- One of the devices is an A32 master, and the memory resides on the other device (the SHARED MEMORY Server device) and is A32 and internally (dual port) accessible.

RULE D.1.8:
A SHARED MEMORY device **SHALL** implement the SHARED MEMORY Connection protocol and the SHARED MEMORY Operation protocol.

RULE D.1.9:
A SHARED MEMORY device **SHALL** support the SHARED MEMORY Synchronous channel type.

RULE D.1.10:
A SHARED MEMORY device **SHALL** implement the *Shared Memory* bit in the VXIbus Protocol Register.

D.2 SERVER REQUIREMENTS

A SHARED MEMORY Server is the SHARED MEMORY device that has control of the shared memory used in a channel. The SHARED MEMORY Server is responsible for performing any configuration of the shared memory required for a particular channel being setup.

RULE D.2.1:
IF a SHARED MEMORY Server uses A24 VMEbus shared memory for its channels,
THEN it **SHALL** implement the 32 bit A24 Pointer Register.

RULE D.2.2:
IF a SHARED MEMORY Server uses A32 VMEbus shared memory for its channels,
THEN it **SHALL** implement the 32 bit A32 Pointer Register.

PERMISSION D.2.1:
A SHARED MEMORY Server device **MAY** use both A24 and A32 shared memory for its channels, but not with the same client.

RECOMMENDATION D.2.1:
To ensure interoperability, an A24 SHARED MEMORY Server device should select all the shared memory for its channels from within the A24 address range 200000_{16} through $DFFFFF_{16}$. All Message Based A24 Masters are required to access this address range. Refer to the Data Transfer Capabilities section of the VXIbus System Specification.

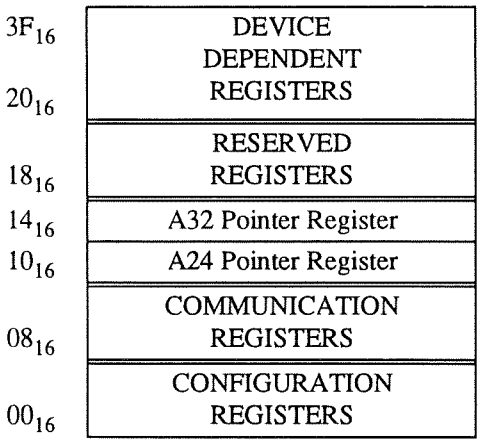
RECOMMENDATION D.2.2:
To ensure interoperability, an A32 SHARED MEMORY Server device should select all the shared memory for its channels from within the A32 address range 20000000_{16} through $DFFFFFFF_{16}$. All Message Based A32 Masters are required to access this address range. Refer to the Data Transfer Capabilities section of the VXIbus System Specification.

The A32 and A24 Pointer registers are used to hold the base address of an area of shared memory that has information needed for connecting SHARED MEMORY channels. This shared memory area is called the Channel Setup Data Area and is described later in this document.

The A24 and A32 Pointer Registers are implemented in one of the following ways.

- Pointer Registers Implemented in A16 Register Space

The location of the A24 and A32 Pointer Registers relative to the Configuration and Communication Registers is shown in the following figure.



RULE D.2.3:
IF the SHARED MEMORY Server uses A24 VMEbus shared memory for its channels
AND it does not implement the A24 Pointer Register in the A16 Register Space,
THEN it **SHALL** return a value of $FFFFFFFF_{16}$ to a read of the A24 Pointer Register location in the A16 register space.

RULE D.2.4:
IF the SHARED MEMORY Server uses A32 VMEbus shared memory for its channels
AND it does not implement the A32 Pointer Register in the A16 Register Space,
THEN it **SHALL** return a value of $FFFFFFFF_{16}$ to a read of the A32 Pointer Register location in the A16 register space.

OBSERVATION D.2.1:
IF a SHARED MEMORY Server does not use A24 VMEbus shared memory,
THEN a SHARED MEMORY Client will not need to read the A24 Pointer Register since the Client will never receive the **Channel Initiate A24** event. The same is also true for the case of A32 VMEbus shared memory.

- Pointer Registers Implemented in A24 Register Space

RULE D.2.5:
IF the Pointer Registers are implemented in the SHARED MEMORY Server A24 Register Space,
THEN the A24 Pointer Register **SHALL** be the first A24 operational register (Offset + 00_{16}) and the A32 Pointer Register **SHALL** be the second A24 operational register (Offset + 04_{16}).

OBSERVATION D.2.2:
The location of a device's A24 operational registers is determined by the Offset Register described in the Configuration Registers section of the VXIbus System Specification.

- Pointer Registers Implemented in A32 Register Space

RULE D.2.6:
IF the Pointer Registers are implemented in the SHARED MEMORY Server A32 Register Space,
THEN the A24 Pointer Register **SHALL** be the first A32 operational register (Offset + 00_{16}) and the A32 Pointer Register **SHALL** be the second A32 operational register (Offset + 04_{16}).

OBSERVATION D.2.3:
The location of a device's A32 operational registers is determined by the Offset Register described in the Configuration Registers section of the VXIbus System Specification.

RECOMMENDATION D.2.3:
A SHARED MEMORY Communication Protocol Server device should implement the pointer registers in A16 Register Space.

The least significant bit (bit 0) of the A24 and A32 Pointer Registers is used to indicate when a read of the Pointer Register will return a valid address.

RULE D.2.7:
IF the value in the A24 Pointer Register is valid,
THEN bit 0 of the A24 Pointer Register **SHALL** be zero (0)
ELSE bit 0 of the A24 Pointer Register **SHALL** be one (1).

RULE D.2.8:
IF the value in the A32 Pointer Register is valid,
THEN bit 0 of the A32 Pointer Register **SHALL** be zero (0)
ELSE bit 0 of the A32 Pointer Register **SHALL** be one (1).

OBSERVATION D.2.4:
The SHARED MEMORY Server should control enough memory to contain all the channels that it has active at any particular time.

D.3 CLIENT REQUIREMENTS

A SHARED MEMORY Client is the SHARED MEMORY device that does not control the shared memory used in a channel.

RULE D.3.1:
A SHARED MEMORY Client **SHALL** have VMEbus master capability.

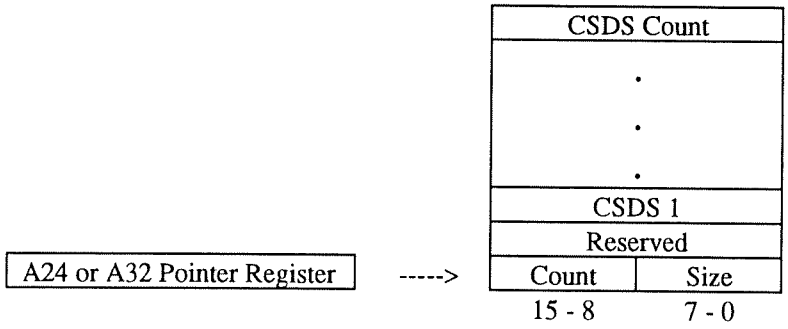
PERMISSION D.3.1:
A SHARED MEMORY Client **MAY** use both A24 and A32 memory for its channels.

E. CHANNEL ESTABLISHMENT

The SHARED MEMORY connection protocol is used by SHARED MEMORY devices to establish a channel between a SHARED MEMORY Server and a SHARED MEMORY Client. The connection protocol does not require that either SHARED MEMORY device be a commander or servant of the other.

E.1 CHANNEL SETUP DATA AREA

The connection protocol relies on a reserved area (Channel Setup Data Area) of the shared memory for passing channel setup information to the SHARED MEMORY Client device. The following figure details the Channel Setup Data Area.



- Count (1 byte): This is the number of Channel Setup Data Structures that are in the area.
- Size (1 byte): This is the size (in bytes) of each Channel Setup Data Structure in the area. This field must have the value 16.
- Reserved (2 bytes): This field is reserved for future use and should default to a value of $FFFF_{16}$. This field also serves to align the data structures onto 4 byte boundaries.
- CSDS 1 (16 bytes): This is the first Channel Setup Data Structure in the area.
- CSDS *Count* (16 bytes): This is the last Channel Setup Data Structure in the area.

RULE E.1.1:
The SHARED MEMORY Server device **SHALL** set up a Channel Setup Data Area with at least one Channel Setup Data Structure. A Channel Setup Data Area **SHALL** be setup for each VMEbus address space used.

RECOMMENDATION E.1.1:
It is recommended that the Channel Setup Data Area have more than one Channel Setup Data Structure. Should a SHARED MEMORY Client device fail before a channel connection has been made, there is a chance that the Channel Setup Data Structure being used could not be reclaimed.

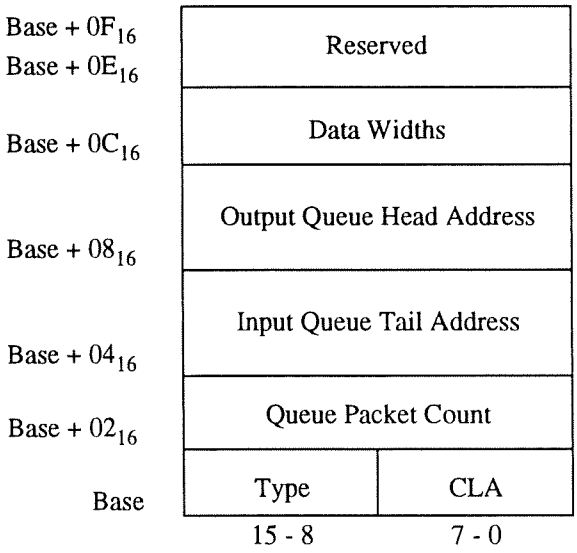
RULE E.1.2:
IF the SHARED MEMORY Server is using A24 memory for its channels,
THEN the A24 Pointer Register **SHALL** contain the A24 address of the beginning of the Channel Setup Data Area for the A24 memory.

RULE E.1.3:
IF the SHARED MEMORY Server is using A32 memory for its channels,
THEN the A32 Pointer Register **SHALL** contain the A32 address of the beginning of the Channel Setup Data Area for the A32 memory.

E.2 CHANNEL SETUP DATA STRUCTURE

RULE E.2.1:
A Server Device **SHALL** fill all fields of the CSDS structure as described below before making the structure available to the Client device.

This is the organization of the Channel Setup Data Structure.



The elements of the Channel Setup Data Structure have the following definitions:

- CLA (1 byte): This is the logical address of the intended SHARED MEMORY Client.

  RULE E.2.2:
  The SHARED MEMORY Server **SHALL** write its logical address in the CLA field of any Channel Setup Data Structure that is not being used.

- Type (1 byte): The Type field indicates the type of SHARED MEMORY channel being established. This field will be 80_{16} for the Synchronous Channel and 81_{16} for the Asynchronous Channel.

- Queue Packet Count (2 bytes): This is the number of queue packets that are in each of the input and output queues.

- Input Queue Tail Address (4 bytes): This field is the address of the tail of the input queue (named with respect to the Server). At initialization, this is also the starting address of the queue. In other words it is the address of the first packet in the input queue.

- Output Queue Head Address (4 bytes): This field is the address of the head of the output queue (named with respect to the Server). At initialization, this is also the starting address of the queue. In other words it is the address of the first packet in the output queue.

- Data Widths (2 bytes): This field specifies the VMEbus data widths supported by the channel memory, in addition to the D16 required by all VXIbus devices. It has the following format:

Reserved	D64	D32B	D32	D08
15 - 4	3	2	1	0

  - D08 (1 bit): A value of one in this field indicates that the channel memory is D08(EO) accessible.

  - D32 (1 bit): A value of one in this field indicates that the channel memory is D32 accessible.

  - D32B (1 bit): A value of one in this field indicates that the channel memory is accessible via the D32 Block Transfer Mode.

  - D64 (1 bit): A value of one in this field indicates that the channel memory is D64 accessible (for VMEbus-64 implementations).

  - Reserved (12 bits): Reserved by the VXIbus consortium and shall always be set to 0.

  RULE E.2.3:
  Each memory location within the channel must be accessible via D16 as well as via each of the indicated data widths.

- Reserved (2 bytes): This field is reserved by the VXIbus consortium and shall always be set to 0.

OBSERVATION E.2.1:
The Channel Setup Data Structure is maintained (by the Server) only during the Connection Protocol. However, the Client must retain the information carried by the Channel Setup Data Structure throughout the channel's operation.

E.3 CONNECTION PROCEDURE

Either a SHARED MEMORY Server or Client can initiate a channel. A SHARED MEMORY Client device initiates a channel by sending one of the **Channel Request** events to the SHARED MEMORY Server device. A SHARED MEMORY Server device can begin a channel connection either when it receives a **Channel Request** event or when it has some internal requirement to initiate a channel connection. A SHARED MEMORY Server device initiates a channel by sending one of the **Channel Initiate** events to the SHARED MEMORY Client device.

RULE E.3.1:
Before attempting to initiate a connection, a SHARED MEMORY device **SHALL** check that the remote device has the *Shared Memory** bit in its Protocol register set to 0.

OBSERVATION E.3.1:
To successfully initiate the SHARED MEMORY Communication Protocol both the Server device and the Client device must have their *Passed* and *Ready* bits set and have all their SHARED MEMORY events enabled.

The SHARED MEMORY connection protocol is described in the following table.

SHARED MEMORY Server	SHARED MEMORY Client
	Send a **Channel Request** event (*if Client initiated*). The Client is not guaranteed that the Server will respond to the **Channel Request**. A time-out is recommended.
IF **Channel Request** event was received and a channel cannot be provided, **THEN** send a **Channel Setup Error** event and do not continue with the connection setup.	
IF a Channel Setup Data Area has not been created, **THEN** create it and load the appropriate shared memory pointer register. Configure a channel and load its parameters into a Channel Setup Data Structure. Send a **Channel Initiate Ann** (nn = 24 or 32) event to the Client.	
Wait for a response. The Server must not modify the Channel Setup Data Structure until it receives either a **Channel Acknowledge** or a **Channel Setup Error** event from the Client.	**IF** **Channel Setup Error** is received, **THEN** do not continue with the connection setup.
	IF **Channel Initiate Ann** is received **THEN** find the Channel Setup Data Structure with the CLA field equal to the Client's logical address, read it, and copy the data to local memory.
	IF unable to complete setup of the channel, **THEN** send a **Channel Setup Error** event and do not continue with the connection setup.
	IF the type of channel indicated in the Type field is not supported, **THEN** send a **Channel Setup Error** event and do not continue with the connection setup. **ELSE** send a **Channel Acknowledge** event and begin using the channel.
IF **Channel Setup Error** is received, **THEN** reclaim the channel and the Channel Setup Data Structure and do not continue with the connection setup.	
IF **Channel Acknowledge** is received, **THEN** the channel has been connected and can now be used. Reclaim the Channel Setup Data Structure.	

TABLE E.1. Connection Protocol

OBSERVATION E.3.2:
A Client device is unable to complete setup of the channel if one of the following conditions occurs:

- The **Channel Initiate** event indicates an address space not supported by the client.
- The Client device cannot access the Channel Setup Data Structure.
- The Channel type specified by the Server device in the CSDS structure is not supported by the Client device.
- The Client device is currently the server for an existing channel between the two devices.

RULE E.3.2:
A SHARED MEMORY capable device that has its *Shared Memory** bit set to 0 **SHALL** always respond to a **Channel Request** or **Channel Initiate** event.

OBSERVATION E.3.3:
The preceding rule ensures that a device that dynamically loads its SHARED MEMORY Communication Protocol driver is capable of returning a **Channel Setup Error** event if the SHARED MEMORY driver is not running at the time a connection request is received.

RULE E.3.3:
A SHARED MEMORY Client and Server pair **SHALL NOT** have more than one connection setup active at one time.

OBSERVATION E.3.4:
The preceding rule prevents any ambiguity with the channel connection setup events that would occur if two connection setups were active simultaneously. It does not prohibit a Server and Client from having both a Synchronous and an Asynchronous channel. The only constraint is that each channel must be completely connected before another connection can be started.

PERMISSION E.3.1:
A Server **MAY** be setting up connections with more than one Client at a time. A Client **MAY** be setting up connections with more than one Server at a time.

RULE E.3.4:
IF a pair of devices establish both a Synchronous channel and an Asynchronous channel,
THEN both channels **SHALL** be in the same address space (A24 or A32).

RULE E.3.5:
IF a pair of devices establish both a Synchronous channel and an Asynchronous channel,
THEN the same device **SHALL** be the Server for both channels.

E.4 EXCEPTION HANDLING

During connection establishment there are a number of exceptions that can occur which are difficult to recover from. This section documents some of these exceptions and offers advice on how to handle them.

During connection setup it is possible for one of the devices involved to cease normal operation, due to device failure, soft reset, receipt of the Abort Normal Operation command, or receipt of the End Normal Operation

command. In addition, one of the devices may take an excessively long time to respond to an event due to some device dependent reason.

When a Client device sends the **Channel Request** event to a Server device, the Server device may be unable to respond to that event.

PERMISSION E.4.1:
IF an appropriate amount of time expires after the Client device has sent the **Channel Request** event,
THEN the Client device **MAY** either cease attempting to request a connection or it **MAY** re-send an identical **Channel Request** event.

RULE E.4.1:
A Client device **SHALL NOT** send **Channel Request** events of different types without receiving an intervening response event.

RECOMMENDATION E.4.1:
The Client device should determine whether the Server device is in the NORMAL OPERATION sub-state before attempting to re-send a **Channel Request** event.

RULE E.4.2:
IF a Server device receives a **Channel Request** event from a Client device to which it has sent a **Channel Initiate** event, but before it has received a **Channel Acknowledge** or **Channel Setup Error** event from that Client device,
THEN the Server device **SHALL** ignore the **Channel Request** event.

RULE E.4.3:
IF a Server device receives multiple **Channel Request** events from a Client device before it has sent a **Channel Initiate** event to that Client,
THEN the Server device shall ignore all but the first **Channel Request** event.

A Client device will receive a **Channel Setup Error** event in response to **Channel Request** events for one of several reasons. If the Server device is not able to set up a channel for the type or address space requested, it will send the **Channel Setup Error** event. If a channel of the type (Synchronous, Asynchronous) being requested already exists between the Client device and the Server device, the Server device will also send the **Channel Setup Error** event.

When a Server device sends a **Channel Initiate** event to a Client device, the Client device may be unable to respond to that event. The Server device may choose to wait forever for a response or it may determine a maximum amount of time to wait.

RECOMMENDATION E.4.2:
IF a Client device does not respond to a **Channel Initiate** event within a reasonable amount of time,
THEN the Server device should abort the setup using the Channel Abort protocol.

RULE E.4.4:
IF a Client device does not respond to a **Channel Initiate** event,
THEN the Server device **SHALL NOT** modify any of the channel resources until it receives the **Channel Setup Error** event, the **Channel Acknowledge** event or it completes the Channel Abort protocol.

RECOMMENDATION E.4.3:
The time that a Server device waits before attempting to abort a channel should be relatively long because a Channel Abort will abort all channels shared with the Client device.

RULE E.4.5:
IF a non-recoverable condition occurs, such as a non-recoverable bus error,
THEN the device detecting the condition **SHALL** initiate the Channel Abort protocol.

During connection setup it is possible for two devices that have both Server and Client capability to send a **Channel Initiate** event to each other simultaneously. This is handled by requiring the device with the numerically smaller logical address to switch to the role of a Client device and continue the connection establishment procedure accordingly.

RULE E.4.6:
IF a Server device that has just sent a **Channel Initiate** event to a Client device receives a **Channel Initiate** event back from that device (which also has Server capability),
THEN the Server device **SHALL** compare its logical address with that of the remote device and perform the following:

> **IF** the logical address of the remote device has a value larger than the Server's own logical address,
> **THEN** the Server device **SHALL** become the Client device for this connection and respond with either the **Channel Acknowledge** event or the **Channel Setup Error** event in accordance with the normal protocol connection procedure.
> **ELSE** the Server device **SHALL** ignore the received **Channel Initiate** event and continue to wait for the appropriate response from the Client device, in accordance with its normal connection procedure.

During connection setup it is also possible for two devices that have both Server and Client capability to send a **Channel Request** event to each other simultaneously. This is handled by requiring the device with the numerically smaller logical address to switch to the role of a Server device and continue the connection establishment procedure accordingly.

RULE E.4.7:
IF a Client device that has just sent a **Channel Request** event to a Server device receives a **Channel Request** event back from that device (which also has Client capability),
THEN the Client device **SHALL** compare its logical address with that of the remote device and perform the following:

> **IF** the logical address of the remote device has a value larger than the Client's own logical address,
> **THEN** the Client device **SHALL** become the Server device for this connection and respond with either the **Channel Initiate** event or the **Channel Setup Error** event in accordance with the normal protocol connection procedure.
> **ELSE** the Client device **SHALL** ignore the received **Channel Request** event and continue to wait for the appropriate response from the Server device, in accordance with its normal connection procedure.

OBSERVATION E.4.1:
Devices that do not support both Client and Server capability satisfy the requirements of the two preceding rules by responding with a **Channel Setup Error** event or ignoring the received event, in accordance with those rules.

F. CHANNEL OPERATION

The SHARED MEMORY Operation protocol defines how communication through a channel is conducted.

F.1 OPERATION OVERVIEW

This section presents an overview of the SHARED MEMORY Operation protocol.

The Operation protocol defines the operation of a shared memory communications channel based on circular queues. A channel can be Synchronous or Asynchronous, defined at the time the channel is created (See Shared Memory Connection Protocol). In a Synchronous channel, there is no requirement that the recipient device handle messages immediately. Asynchronous channels are used where high priority communication is to occur. Examples include warnings, events, asynchronous reports ("Acquisition Complete"), and commands which must be executed immediately, such as status requests and emergency shutdown commands. The receiving Higher Layer Entity is expected to extract that message from the SHARED MEMORY layer as quickly as possible.

The Operation Protocol presently defines four handshaking events for coordinating transfers through the shared memory: two events for Synchronous channels and two events for Asynchronous channels.

F.1.1 Data Structures Used

The SHARED MEMORY Operation Protocol defines seven types of data structures which are used in the protocol: buffers, buffer headers, buffer data areas, buffer lists, lists of buffer lists, queues, and command packets. The Operation Protocol also defines certain event-codes that are sent between devices via VXIbus signals or interrupts to synchronize data transfers.

F.1.2 Transferring Data

Data is transferred between devices in buffers which are established in the shared memory. The data is placed into shared memory buffers by the sending side and the receiving side removes that data as needed/desired.

F.1.3 Using Buffers

Buffers contain the data passed between the two communicating SHARED MEMORY Communication Protocol devices. A Buffer consists of a header and a data area. Both the header and data area are described in more detail later in this document. Buffers are bound together into linked lists. The data-sending side acquires buffers, fills those buffers, and sends the buffer list to the data-receiving side through the shared memory. Lists of buffers may themselves be acquired and returned in lists, for improved performance.

F.1.4 Using Packets

A packet contains control information which is used to manage the communications through the SHARED MEMORY channel. Such a packet contains commands which, for example, request a given number of buffers, return a given number of buffers, or return a buffer list containing data.

These packets are built by one SHARED MEMORY device and sent to the other through the shared memory.

F.1.5 Using Queues

The shared memory queues are used to implement an abstract communication system consisting of unidirectional streams, through which the packets (which themselves contain the commands that control the channel) are sent from one device to the other. Each channel has an associated pair of queues: an Input queue and an Output queue (named with respect to the Server.) A Server accepts SHARED MEMORY Communication Protocol commands via the Input queue, and provides responses via the Output queue. A Client accepts SHARED MEMORY Communication Protocol commands via the Output queue, and places responses in the Input queue. The queue used by a device to send commands is its *send queue*. The queue used by a device to receive commands is its *receive queue*. The input queue is thus the Client's send queue and the Server's receive queue. The output queue is the Server's send queue and the Client's receive queue. Packets contain the addresses of shared memory buffers, as needed.

F.2 BUFFERS

The SHARED MEMORY Operation protocol transmits data between communicating SHARED MEMORY devices by placing it into linked lists of shared memory buffers.

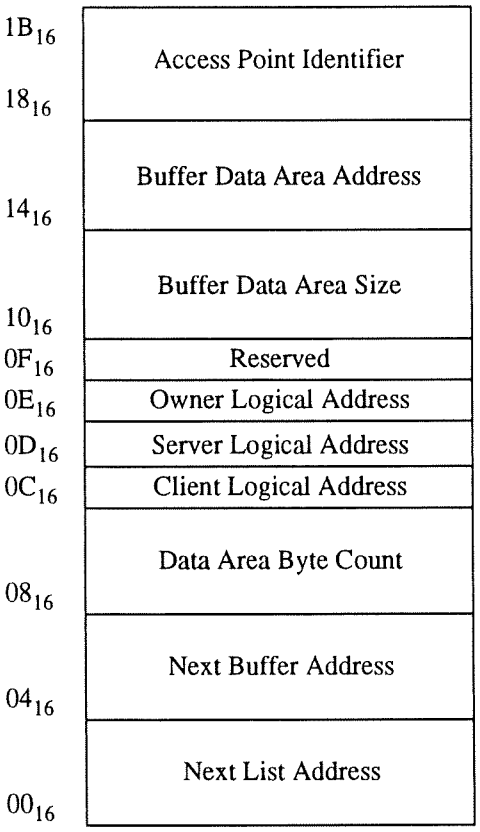
A Buffer consists of a Buffer Header and an associated Buffer Data Area pointed to by the Buffer Header. To allow for memory resource management the Client device always treats a Buffer as a single unit. This means that the Buffer Header always points to the same Buffer Data Area until the Buffer is returned to the Server device. A Buffer address always points to the base of the Buffer Header.

RULE F.2.1
When a SHARED MEMORY Client device returns a Buffer to the Server device, the Buffer **SHALL** consist of the same Buffer Header and Buffer Data Area as when it was received by the Client device.

F.2.1 Buffer Header

A Buffer Header contains all the buffer control information required by the SHARED MEMORY Communication Protocol.

RULE F.2.2:
A SHARED MEMORY Communication Protocol Buffer Header **SHALL** be a contiguous data structure, 28 bytes in length, and contain the following fields:



- Next List Address (4 bytes): This field contains the address of the next buffer list.

- Next Buffer Address (4 bytes): This field contains the address of the next buffer in the current buffer list.

- Data Area Byte Count (4 bytes): This field defines the number of bytes of data written in the buffer data area and does not include the size of the buffer header.

OBSERVATION F.2.1:
The sending device may choose to increment the Data Area Byte Count field as bytes are written into the buffer, or write the total number of bytes once all the bytes have been written.

- Client Logical Address (1 byte): This field defines the logical address of the Client device participating in the channel using this buffer.

- Server Logical Address (1 byte): This field defines the logical address of the Server device participating in the channel using this buffer.

- Owner Logical Address (1 byte): This field defines the logical address of the device which owns the physical memory of the buffer.

- Reserved (1 byte): This field is reserved for use by the VXIbus consortium.

- Buffer Data Area Size (4 bytes): This field contains the total size (in bytes) of the Buffer Data Area.

- Buffer Data Area Address (4 bytes): This field contains the address of the Buffer Data Area associated with this buffer.

- Access Point Identifier (4 bytes): This field defines the Access Point Identifier for the Higher Layer Entity receiving the data. Bit 31 is used to differentiate between VXIbus Reserved Access Point Identifiers and user defined Access Point Identifiers, as shown below:

Bit #:	31	30 - 0
Contents:	1	VXIbus Reserved Access Point Identifier

Bit #:	31	30 - 12	11 - 0
Contents:	0	Manuf. Access Point Identifier	Manuf. ID

The Manufacturer ID field in user defined Access Point Identifiers contains the VXIbus Manufacturer ID of the manufacturer who originally defined the Higher Layer Entity to be accessed through the Access Point Identifier. This may be different from the manufacturer of the device implementing the protocol.

F.2.2 Buffer Data Area

The Buffer Data Area contains the data being transmitted by the SHARED MEMORY Buffer.

RULE F.2.3:
The length of the Buffer Data Area **SHALL** be at least 256 bytes and **SHALL** be a multiple of four bytes.

RULE F.2.4:
The Buffer Data Area **SHALL** be aligned on a long word boundary.

OBSERVATION F.2.2:
The format of the data within a Buffer Data Area is APID dependent.

F.2.3 Buffer Lists

A Buffer List is defined as a linked list of buffers.

RULE F.2.5:
The Next Buffer Address field of each Buffer **SHALL** contain the address of the next Buffer in the Buffer List.

RULE F.2.6:
The terminating Buffer in a Buffer List **SHALL** have a Next Buffer Address value of $FFFFFFFF_{16}$.

F.2.4 Lists of Buffer Lists

A list of buffer lists is defined as a linked list of buffer lists. A list of buffer lists is used for exchanging multiple lists of unused buffers between devices in a single transfer, for improved performance. It may also be used by the SHARED MEMORY Communication Protocol for managing multiple lists of data buffers as they are received by a device prior to exporting them to a Higher Layer Entity.

RULE F.2.7:
The Next List Address field of the first buffer in each buffer list **SHALL** contain the address of the first buffer in the next buffer list in the list of buffer lists. The Next List Address field of all other buffers **SHALL** be assigned the value $FFFFFFFF_{16}$.

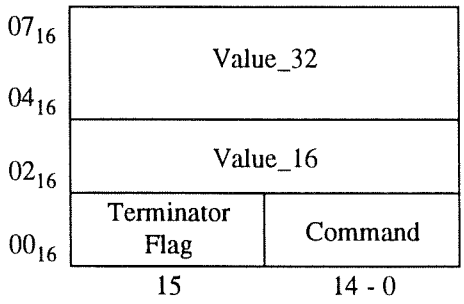
RULE F.2.8:
The first buffer of the last list in a list of buffer lists **SHALL** have a Next List Address value of $FFFFFFFF_{16}$.

OBSERVATION F.2.3:
The first buffer in a single list of buffers has a Next List Address value of $FFFFFFFF_{16}$.

F.3 PACKETS

Operation Protocol commands are passed in queue packets, via the packet handling protocol.

RULE F.3.1:
A Command Packet **SHALL** be 8 bytes in length, and contain the following fields, as illustrated in the diagram below:



- Command (15 bits): This field is used for packet commands. All command field values are reserved for use by the SHARED MEMORY Communication Protocol.

- Terminator Flag (1 bit): This field is used to indicate the presence (value 0) or absence (value 1) of a valid command in the packet. Its use is described in the packet handling section.

- Value_16 (2 bytes): The Value_16 field holds any 16-bit parameter that is required with a packet command.

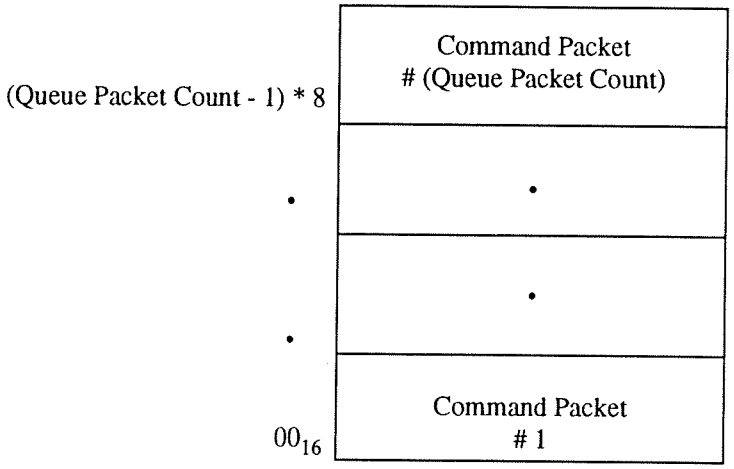
- Value_32 (4 bytes): The Value_32 field holds any 32-bit parameter that is required with a packet command.

A device making a request of the other device on the channel does so by filling the command field with the appropriate command value and loading the Value_16 and/or Value_32 fields as needed for that command. This is then followed by setting the Terminator Flag to 0.

F.4 QUEUES

Queues contain ordered lists of command packets. A queue is a circular buffer of command packets and has both a head and a tail address. Queues are used to transfer command packets between devices. Because of the nature of the queues and the way packets are placed into and taken out of the queues, two devices can reliably communicate without additional hardware such as read-modify-write support.

Queue Memory Organization



RULE F.4.1:
The number of Command Packets in a Queue of a given SHARED MEMORY channel **SHALL** equal the Queue Packet Count field of the Channel Setup Data Structure for that channel.

RULE F.4.2:
All queues **SHALL** contain two or more command packets.

RULE F.4.3:
All SHARED MEMORY Server devices **SHALL** initialize their Queue pointers to the following state before data transfers commence:

- The Input Queue HEAD address **SHALL** contain the address of input Command Packet entry #1.
- The Output Queue TAIL address **SHALL** contain the address of output Command Packet entry #1.
- The Terminator Flag field **SHALL** equal one (1) in all Command Packets in the Queue.

RULE F.4.4:
All SHARED MEMORY Server devices **SHALL** initialize the Input Queue TAIL Address entry in the Channel Setup Data Structure to contain the address of the input Command Packet entry #1.

RULE F.4.5:
All SHARED MEMORY Server devices **SHALL** initialize the Output Queue HEAD Address entry in the Channel Setup Data Structure to contain the address of the output Command Packet entry #1.

OBSERVATION F.4.1:
The Input Queue TAIL address and Output Queue HEAD address are passed to the Client in the Channel Setup Data Structure at channel creation time. These pointers are needed only by the Client, and should be maintained by the Client.

RULE F.4.6:
The Server device **SHALL** write Command Packets into the Output Queue and the Client device **SHALL** read Command Packets from the Output Queue.

RULE F.4.7:
The Client device **SHALL** write Command Packets into the Input Queue and the Server device **SHALL** read Command Packets from the Input Queue.

OBSERVATION F.4.2:
The queues are named with respect to their function from the perspective of the Server.

During operation, both the HEAD and TAIL address "wrap-around" when they reach the end of the Queue. The addresses are reset to point at the first packet entry in the Queue. The wraparound point and address reset positions can be determined and recorded at channel initiate time by examining the Channel Setup Data Structure (See the Connection Protocol).

F.5 CHANNEL OPERATION

The normal operation of a SHARED MEMORY Communication Protocol channel consists of one of the devices acquiring a buffer list, filling the buffers with data and making them available to the other device. If the receiving device is not the original owner of the buffers it returns the buffer list to the owner. This cycle may be repeated by either device until an exception condition occurs or one of the devices terminates the channel. During normal operation, certain error conditions may arise and must be handled by the devices.

F.5.1 Packet Handling

Packets are used to transfer control information between devices during channel operation as well as for the purpose of channel termination. The sending device fills the first available packet entry in the appropriate queue with the command it wishes to transfer to the receiving device. The sending device also fills in the parameter values required by that command. The receiving device may then access the queue and retrieve the command from the packet.

RULE F.5.1:
The Command Packet Write operation **SHALL** follow the steps presented below when writing a command to the send queue:

- The writing device **SHALL** test the Terminator Flag field of the Command Packet entry referenced by the Queue TAIL address.
  IF the Terminator Flag is zero (0),
  THEN the Queue is full, and the writing device **SHALL** stop the Command Packet Write Operation until a **Queue Not Full Synchronous** event or a **Queue Not Full Asynchronous** event is received,
  ELSE the device **SHALL** proceed with the following steps, in order:

- The writing device **SHALL** write the contents of the Command, Value_16 and Value_32 fields of the Command Packet.

- The writing device **SHALL** then set to zero (0) the Terminator Flag field of the Command Packet referenced by the Queue TAIL address, to mark the Packet as in use.

- The writing device **SHALL** test the Terminator Flag field of the Command Packet previous to the Command Packet entry referenced by the Queue TAIL address.
  IF the Terminator Flag equals one (1),
  THEN the writing device **SHALL** send a **Queue Not Empty Synchronous** or a **Queue Not Empty Asynchronous** event, depending on the type of the queue.

- The writing device **SHALL** advance the TAIL address to point at the next Command Packet entry.
  IF the TAIL address was pointing at Command Packet entry #n, where "n" equals the number of packet entries in the queue *(Queue Packet Count)*,
  THEN the TAIL address **SHALL** be set to point to Command Packet entry #1.

RULE F.5.2:
The Command Packet Read operation **SHALL** follow the steps presented below when reading a command from the receive queue:

- The reading device **SHALL** test the Terminator Flag field of the Command Packet entry referred to by the Queue HEAD address.
  IF the Terminator Flag equals one (1),
  THEN the Queue is empty, and the reading device **SHALL** stop the Command Packet Read operation until a **Queue Not Empty Synchronous** event or a **Queue Not Empty Asynchronous** event is received,
  ELSE the device **SHALL** proceed with the following steps, in order:

- The reading device **SHALL** read the contents of the Command, Value_16 and Value_32 fields of the Command Packet.

- The reading device **SHALL** then set to one (1) the Terminator Flag field of the Command Packet referenced by the Queue HEAD address, to mark the Packet as available.

- The reading device **SHALL** test the Terminator Flag field of the Command Packet previous to the Command Packet entry referenced by the Queue HEAD address.
  IF the Terminator Flag equals zero (0),
  THEN the reading device **SHALL** send a **Queue Not Full Synchronous** or a **Queue Not Full Asynchronous** event, depending on the type of the queue.

- The reading device **SHALL** then advance the HEAD address to point to the next Command Packet entry.
  IF the Queue HEAD address was pointing at Command Packet entry #n, where "n" equals the number of Packet entries in the queue *(Queue Packet Count)*,
  THEN the Queue HEAD address **SHALL** be set to point at Command Packet entry #1.

OBSERVATION F.5.1:
Receiving a **Queue Not Empty Synchronous** event or a **Queue Not Empty Asynchronous** event does not guarantee that the queue is actually not empty. This is due to the unsynchronized behavior of the sending device and the receiving device. The receiving device may already have read the Command Packet that caused the event to be sent.

OBSERVATION F.5.2:
Receiving a **Queue Not Full Synchronous** event or a **Queue Not Full Asynchronous** event does not guarantee that the queue is actually not full. This is due to the unsynchronized behavior of the sending device and the receiving device. The device receiving the event may already have written additional Command Packets to cause the queue to be full again.

In order to prevent any deadlock situation resulting from both the send and receive queues being full at the same time, it is important that devices give priority to handling incoming commands over generating new outgoing commands.

RULE F.5.3:
When a SHARED MEMORY device finishes writing each command packet to its send queue:
IF that device's receive queue contains any valid command packets,
THEN the device **SHALL** read and process at least one command packet from its receive queue before writing any new command packets to its send queue.

OBSERVATION F.5.3:
Processing the received command packet may itself require responding with another command packet. The above rule requires this response packet to be written to the send queue before writing any other command packets.

F.5.2 Packet Error Handling

This section defines the handling of SHARED MEMORY Communication Protocol errors. There are three types of detectable protocol errors. The first is the receipt of a packet containing an unrecognizable command. The second error is the presence of an invalid memory address referencing a buffer header or data area. The third error is the receipt of an unrecognized Access Point Identifier.

F.5.2.1 UNRECOGNIZED COMMANDS

The UNKNOWN_COMMAND packet is used to indicate the receipt of an unrecognized command. This packet includes the command causing the error.

RULE F.5.4:
IF a device receives a packet containing a command not implemented by that device,
THEN it **SHALL** respond with the UNKNOWN_COMMAND packet having the Value_16 field containing the unrecognized command.

F.5.2.2 INVALID MEMORY ADDRESSES

The INVALID_ADDR packet is used to indicate the presence of an invalid memory address field within either a received packet or a received buffer header. An address is invalid if it references an address that either references memory inaccessible to the device or is improperly aligned for the referenced data structure (such as an odd address). Such an address may be encountered in a packet as a buffer list address, in a buffer header's next buffer address field, or in a buffer header's data area address field. The INVALID_ADDR packet indicates which of these fields contains the invalid address along with either the address itself (if encountered in a packet), or the address of the buffer header containing the invalid address.

RULE F.5.5:
IF a device detects an invalid address in a received packet or buffer header,
THEN it **SHALL** respond with an INVALID_ADDR packet.

F.5.2.3 UNRECOGNIZED ACCESS POINT IDENTIFIER

The UNKNOWN_APID packet is used to indicate that the sending device has received a buffer list with one or more buffers containing an Access Point Identifier which it does not recognize. The Value_32 field contains the Access Point Identifier in error.

RULE F.5.6:
IF a device receives one or more buffers with an Access Point Identifier which it does NOT recognize,
THEN the device **SHALL** send an UNKNOWN_APID packet to the sending device.

F.5.3 Buffer Management

When a connection is set up, the Server device owns all the buffers. This is marked in each buffer header's Server Logical Address field. All buffers are free (i.e. unused) and are placed in the server's free buffer pool. Accordingly the server can immediately fill one of the free buffers with data and send it to the client.

When the connection is set up, the Client device has no buffers. Accordingly it must first acquire a free buffer from the Server device before it can fill it with data and send it back to the server. It is implementation dependent when the Client device acquires free buffers from the Server device. The Client device can implement its own free buffer pool and place the acquired free buffers in it for later use, or it can acquire free buffers from the Server device on demand.

RULE F.5.7:
Prior to allocating a buffer to a SHARED MEMORY channel, the Server device **SHALL** appropriately fill out the Owner Logical Address, Server Logical Address, Client Logical Address, Buffer Data Area Address and Buffer Data Area Size fields of the buffer header.

F.5.3.1 ACQUIRING BUFFERS

A device acquires free buffers from another device by sending a GET_MEM or a GET_MEM_WAIT packet to that device. The Value_16 and the Value_32 fields contain the number and the size of the buffer lists the device wants to acquire.

A client will need to acquire free buffers when it needs to either fill its own free buffer pool, or when it needs to send data to the server.

A server will need to acquire free buffers when the server wants to send data to the client and has no free buffers in its free buffer pool. If the client does not return buffers when the data in them are read, but instead places the buffers in its own free buffer pool the server's free buffer pool will eventually be empty. The server can then reclaim some buffers from the client.

A device may need to acquire more than one buffer list at a time. Instead of sending multiple requests it can specify a number of buffer lists greater than one.

The size of the buffer list is the total size of all the buffer data areas in the buffer list.

The GET_MEM packet command results in an immediate response which may return no free buffers. The GET_MEM_WAIT packet command will not return a response until at least one free buffer is available.

RULE F.5.8:
IF a device receives a GET_MEM packet
THEN the device **SHALL** perform the following steps:

- The device **SHALL** build buffer lists from its free buffer pool. The total size of the buffer data areas in each buffer list should at least equal the amount requested in the Value_32 field in the GET_MEM packet. The device should build as many buffer lists as indicated in the Value_16 field of the GET_MEM packet.

- The device **SHALL** build a list of buffer lists by using the Next List Address field.

- **IF** the device's free buffer pool does not contain a sufficient number of buffers to satisfy the requested buffer list size or number of buffer lists,
  THEN the device SHALL build as many buffer lists as possible, where the last buffer list may be less than the requested size.

- The device **SHALL** send a MEM_AVAIL packet to the device it received the GET_MEM packet from. The Value_32 field of the packet contains the address of the first buffer in the first buffer list. A value of $FFFFFFFF_{16}$ indicates that the field contains no buffer list address. The Value_16 field of the packet contains the status from the handling of the request. A value of 0 indicates that the request was fully satisfied. A value of 1 indicates that the request was not fully satisfied. A value of 3 indicates that the request can never be satisfied due to limited number/size of buffers that are available to the channel.

RULE F.5.9:
IF a device receives a GET_MEM_WAIT packet
THEN the device SHALL perform the following steps:

- The device **SHALL** build buffer lists from its free buffer pool. The total size of the buffer data areas in each buffer list should at least equal the amount requested in the Value_32 field in the GET_MEM_WAIT packet. The device should build as many buffer lists as indicated in the Value_16 field of the GET_MEM_WAIT packet.

- The device **SHALL** build a list of buffer lists by using the Next List Address field.

- **IF** the device's free buffer pool does not contain a sufficient number of buffers to satisfy the requested buffer list size or number of buffer lists,
  THEN the device SHALL build as many buffer lists as possible, where the last buffer list may be less than the requested size.

- **IF** the device's free buffer pool does not contain any buffers,
  THEN the device **SHALL** wait until at least one buffer is available in the free buffer pool and then build as many buffer lists as possible.

- After assembling the buffer lists, the device **SHALL** send a MEM_AVAIL packet to the device it received the GET_MEM_WAIT packet from. The Value_32 field of the packet contains the address of the first buffer in the first buffer list. The Value_16 field of the packet contains the status from the handling of the request. A value of 0 indicates that the request was fully satisfied. A value of 1 indicates that the request was not fully satisfied. A value of 3 indicates that the request can never be satisfied due to limited number/size of buffers that are available to the channel.

- **IF** the device has a pending GET_MEM_WAIT packet from another device and it receives a GET_MEM or a GET_MEM_WAIT packet from the same device,
  THEN it **SHALL** send an UNEXPECTED_PACKET packet to the device. The Value_16 field of the packet contains the GET_MEM or the GET_MEM_WAIT command value.

RULE F.5.10:
IF a device has not yet received the MEM_AVAIL packet as a response to a GET_MEM_WAIT packet,
THEN the device **SHALL NOT** send further GET_MEM or GET_MEM_WAIT packets to the same device.

RULE F.5.11:
When a device receives a GET_MEM_WAIT packet which it cannot immediately satisfy, it **SHALL** continue to process other packets while waiting for a free buffer to become available.

RULE F.5.12:
After a device sends a GET_MEM or a GET_MEM_WAIT packet, it **SHALL** continue to process other packets while waiting for the resulting MEM_AVAIL packet.

OBSERVATION F.5.4:
Receiving a DATA_AVAIL command does not cancel an outstanding GET_MEM_WAIT command.

RULE F.5.13:
A client **SHALL NOT** change the content of the Owners Logical Address, Server Logical Address, Client Logical Address, Buffer Data Area Size or Buffer Data Area Address fields of the buffers it has acquired.

RULE F.5.14:
A Client device **SHALL NOT** transfer a buffer to any device other than the Server device indicated in the buffer's Server Logical Address field.

OBSERVATION F.5.5:
It is implementation dependent what action the client takes when it receives a MEM_AVAIL packet. If the client implements its own free buffer pool it may place the buffer list in the free buffer pool. If it acquires buffers on demand it may fill the buffers with data and immediately send the buffer list back to the server.

PERMISSION F.5.1:
When a Client device receives a MEM_AVAIL packet it **MAY** rearrange the buffers in the buffer lists pointed to by the Value_32 field of the packet in any way the Client pleases.

OBSERVATION F.5.6:
A Client device that does not implement its own free buffer pool is considered to have an empty free buffer pool.

F.5.3.2 SENDING DATA

A device sends a buffer list filled with data by sending a DATA_AVAIL packet to another device. The Value_32 field contains the address of the first buffer in the buffer list.

RULE F.5.15:
When a device sends a DATA_AVAIL packet, all the Access Point Identifier fields in the buffer list **SHALL** contain the same access point identifier value.

RULE F.5.16:
When a device sends a DATA_AVAIL packet, all the Next List Address fields in the buffer list **SHALL** contain the value $FFFFFFFF_{16}$.

RULE F.5.17:
When a device receives a DATA_AVAIL packet, it **SHALL** test the Access Point Identifier field of a buffer in the buffer list.
IF the Access Point Identifier field identifies an access point that is registered at the device
THEN the device **SHALL** deliver the buffer list to the access point,
ELSE the device **SHALL** return the buffer list to a free buffer pool and send an UNKNOWN_APID packet to the sending device. The Value_32 of the packet contains the Access Point Identifier value that was unknown to the device.

OBSERVATION F.5.7:
After the receiving device has processed the data in the buffer list it received with the DATA_AVAIL packet, the buffer list is returned to a free buffer pool.

OBSERVATION F.5.8:
The value of the Access Point Identifier field of a buffer has to be valid only when the buffer is sent to another device with the DATA_AVAIL packet.

F.5.3.3 RETURNING BUFFERS

Clients can return buffers to the server at any time without having received any GET_MEM or GET_MEM_WAIT packets. It is implementation dependent when a client decides to return buffers. A client that does not implement its own free buffer pool returns buffers as soon as it has read the data in a buffer list it has received with a DATA_AVAIL packet. A client that implements its own free buffer pool returns buffers when its free buffer pool is overfilled.

RULE F.5.18:
IF a Client wishes to return buffers to the Server device without having received a GET_MEM or a GET_MEM_WAIT packet
THEN the Client device **SHALL** send a MEM_AVAIL packet to the server. The Value_32 field of the packet contains the address of the first buffer in the list of buffer lists being returned. The Value_16 field contains the value 4. All the buffers in the list of buffer lists **SHALL** be owned by the Server device the packet is sent to.

F.5.4 Example: Client Writes to Server

A Client sending data to a Server device via the operation protocol first adds a command packet with a GET_MEM command to the particular queue desired. Whether the Synchronous or Asynchronous channel is chosen depends upon how time-critical the Client considers the message to be: is it worth the interruption of the Server's application to service the Asynchronous request or not?

The Server device reads the GET_MEM request from its input queue, and tries to extract shared memory buffer lists from its current free list of buffers. The Server device attempts to satisfy the Client requirements for the minimum buffer set and minimum number of bytes per buffer set. If these requirements cannot be satisfied, the Server device returns as many buffers as it can afford. The address of the first buffer in this buffer list is sent in the MEM_AVAIL packet in the corresponding output queue.

The Client reads the MEM_AVAIL command from the packet and writes its data into the buffer list. If there is no buffer list, then the Client must issue a new GET_MEM command before any data can be transferred. Once the data is in the buffer list, the address of the first buffer in the list is written with a DATA_AVAIL packet in the desired queue. If the memory supplied in the MEM_AVAIL command was insufficient, the Client device issues another GET_MEM command.

The Server reads the DATA_AVAIL command and extracts the shared memory buffer address from the packet and uses this address to extract the data from the buffer list. The now-empty buffer list is returned to the free buffer pool.

F.5.5 Example: Server Writes to Client

The Server gets a buffer list and fills it with data. The address of the first buffer in this buffer list is then copied into a DATA_AVAIL packet and written in the desired output queue. The Client extracts the contents of the shared memory buffer list and writes the now-empty buffers, along with a MEM_AVAIL command, into the corresponding input queue. The Server reads the MEM_AVAIL command and returns the empty shared memory buffers to the free pool.

F.6 CHANNEL UTILITIES

Two utilities not directly related to data transfer are provided by the SHARED MEMORY Communication Protocol. These are the Channel Test protocol and Access Point Identifier resolution.

F.6.1 Channel Test

The Channel Test protocol allows a SHARED MEMORY device to check if the SHARED MEMORY Communication Protocol connection to the remote device is functioning. The inquiring device sends the TEST command packet to the remote device. The device includes a 48-bit unique sequence number in both the Value_32 and Value_16 fields of the command packet. When the command is received by the remote device it immediately responds with a TEST_ACK packet with the Value_32 and Value_16 fields containing the same values received in the TEST packet. The original sending device can then match the TEST_ACK command to the corresponding TEST command.

RULE F.6.1:
When a device receives a TEST command packet it **SHALL** immediately respond with a TEST_ACK command packet with the Value_32 field and Value_16 field both containing the same values as those received in the TEST command.

F.6.2 Access Point Identifier Resolution

An Access Point Identifier uniquely identifies a Higher Layer Entity to the SHARED MEMORY Communication Protocol. The Access Point Identifier Resolution protocol allows a SHARED MEMORY device to inquire if a Higher Layer Entity with a particular Access Point Identifier is available at the remote device.

A device wishing to inquire about a particular Access Point Identifier sends the APID_QUERY command packet. The Value_32 field contains the Access Point Identifier. Upon receipt of this command, the receiving device responds with the APID_SUPPORTED or APID_NOT_SUPPORTED command. In either case, the Value_32 field in the response contains the same Access Point Identifier as the one received in the APID_QUERY command packet.

RULE F.6.2:
IF a SHARED MEMORY device supports the Access Point Identifier it receives in an APID_QUERY command packet,
THEN the device **SHALL** respond with an APID_SUPPORTED command packet containing the Access Point Identifier in the Value_32 field,
ELSE the device **SHALL** respond with an APID_NOT_SUPPORTED command packet containing the Access Point Identifier in the Value_32 field.

F.6.3 Address Range Resolution

The Address Range Resolution protocol allows a Server device to determine the range of VMEbus addresses a Client device can access. This capability allows a Server device to determine if a Client device can access a block of memory under the Server device's control that is located outside the range required for VXIbus Bus Masters (200000_{16} to $DFFFFF_{16}$ for A24 and 20000000_{16} to $DFFFFFFF_{16}$ for A32).

The Server device sends an ADDR_RANGE command packet to the Client device. The Client device will embed the upper and lower limits of its address range in an ADDR_RANGE_ACK command packet and send this response to the Server device.

RULE F.6.3:
IF a SHARED MEMORY Client device receives an ADDR_RANGE command packet,
THEN the Client device **SHALL** respond with an ADDR_RANGE_ACK command packet. The Value_16 field contains the most significant 16 bits of the lowest VMEbus address accessible by the Client device. The least significant word of the Value_32 field contains the most significant 16 bits of the first VMEbus address after the highest VMEbus address accessible by the Client device.

OBSERVATION F.6.1:
For A24 channels, the physical VMEbus address is computed by left-shifting the 16-bit value 8 places and filling the 8 least significant bits of the A24 address with zeros. For A32 channels, the physical VMEbus address is computed by left-shifting the 16-bit value 16 places and filling the 16 least significant bits of the A32 address with zeros.

OBSERVATION F.6.2:
The value computed for the upper bound for an address range actually represents the first address after the accessible address range. If the highest address in the address space is supported, the Value_32 field is set to 0.

RULE F.6.4:
A SHARED MEMORY Server device **SHALL** by default allocate all of its shared channel resources in the VMEbus address range required for VXIbus Bus Masters. Prior to using any shared channel resources located outside the VMEbus address range required for VXIbus Bus Masters, the Server device **SHALL** ensure that the Client device can access the shared channel resources.

F.7 CHANNEL ABORT

The Channel Abort protocol should be used when it is necessary to cease usage of a SHARED MEMORY Communication Protocol channel quickly without using the resources of that channel. Such a need could arise if a bus error is encountered during either channel setup or channel operation. There may be other events that also necessitate the use of this protocol.

RULE F.7.1:
IF a SHARED MEMORY device encounters a non-recoverable event (which is not handled by any other SHARED MEMORY Communication Protocol) during channel operation or setup with another SHARED MEMORY device,
THEN it **SHALL** cease operation or setup of that channel and any other channel shared with the same device. It **SHALL** also send a **Channel Abort** event to the device with which it shares the channel.

RULE F.7.2:
IF a **Channel Abort** event is received by a SHARED MEMORY client device
THEN the client device **SHALL** cease access to all channel resources or channel setup data structures associated with the server sending the **Channel Abort** event and the Client device **SHALL** send a **Channel Abort Ack** event back to the Server device.

PERMISSION F.7.1:
IF a Server device receives a **Channel Abort** or **Channel Abort Ack** event during the operation or setup of a channel,
THEN it **MAY** re-use any SHARED MEMORY resources associated with the device sending the **Channel Abort** event.

PERMISSION F.7.2:
A new channel setup **MAY** be attempted after a channel operation or setup has been aborted between two devices.

OBSERVATION F.7.1:
A channel may be aborted due to failed memory hardware. It would be prudent to verify that the channel resources are in working order before attempting to re-use them.

RECOMMENDATION F.7.1:
When a device receives an *Abort Normal Operation* Word Serial Command or a *Soft Reset*, it should attempt to abort all channels which it has active or is in the process of setting up.

F.8 CHANNEL TERMINATION

The Channel Termination protocol gracefully closes the specified channel. It insures that no data is lost before the Channel is closed and deallocated. The Channel Termination protocol operates separately on Synchronous and Asynchronous channels. Either the Shared Memory Server or Client can begin the Channel Termination. The packet command sequence is different, depending on whether the Server or Client initiates the Channel Termination.

When the server initiates the termination protocol, it proceeds as follows:

- The SHARED MEMORY Server places the CLOSE command packet on its Output Queue. The Server sends no further packets to the Client.
- The Client receives the CLOSE packet and sends the CLOSE_RET command packet to the Server. The Value_32 field of the packet contains the address of a Buffer List. This Buffer List contains all Buffers that the Client was using with this Channel.
- The Server receives the CLOSE_RET packet. The Server can now reclaim the Queue and Buffer resources.

When the client initiates the termination protocol, it sends the CLOSE_REQ packet to the server. The protocol then proceeds as if the server initiated the termination protocol.

RULE F.8.1:
When a SHARED MEMORY Client initiates the Channel Termination protocol, it **SHALL** follow the following steps:

- Write the CLOSE_REQ command packet to the Server's Input Queue. Send no other packets to the Server (aside from the CLOSE_RET packet specified below.)
- Wait for a CLOSE command packet to arrive from the server.
- Upon receipt of the CLOSE command packet, send the CLOSE_RET command packet to the Server device and cease access to all channel resources.

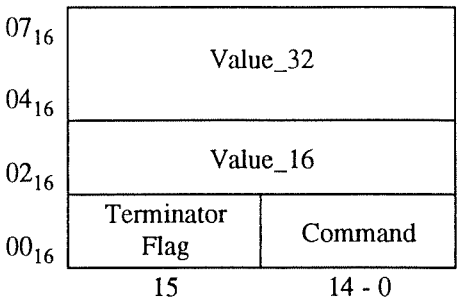
RULE F.8.2:
When a SHARED MEMORY Client device receives an unsolicited CLOSE command packet, it **SHALL** send the CLOSE_RET command packet and cease access to the channel resources.

RULE F.8.3:
When a server initiates the Channel Termination protocol or receives a CLOSE_REQ command packet from a client, it **SHALL** execute the following sequence:

- Place the CLOSE command packet on the Output Queue. Send no further packets to the Client.
- Wait for a CLOSE_RET packet from the client. While waiting, ignore any CLOSE_REQ packets received from the client.
- Upon receipt of the CLOSE_RET packet, the Queue and Buffer resources may be reclaimed.

G. PACKET COMMANDS

All packet commands have the following format:



RULE G.1.1:
All SHARED MEMORY devices **SHALL** implement the following packet commands:

Command	Meaning
GET_MEM	**Command Value: 0001_{16}.** Informs the receiving device that this is a request for shared memory buffers. This command requires an immediate MEM_AVAIL response whether or not there are any buffers available. **Value_16:** Contains a 16-bit unsigned integer representing the minimum number of buffer lists being requested. **Value_32:** Contains a 32-bit unsigned integer representing the minimum number of bytes needed for each buffer list.
GET_MEM_WAIT	**Command Value: 0011_{16}.** Informs the receiving device that this is a request for shared memory buffers. If the receiving device has no shared memory buffers available it must wait until it has at least one buffer before sending the MEM_AVAIL command. **Value_16:** Contains a 16-bit unsigned integer representing the minimum number of buffer lists being requested. **Value_32:** Contains a 32-bit unsigned integer representing the minimum number of bytes needed for each buffer list.
MEM_AVAIL	**Command Value: 0002_{16}.** Sends a list of unused buffer lists. This may be a response to a GET_MEM command, a response to a GET_MEM_WAIT command or it may be a device simply returning buffers, unsolicited. **Value_16:** Indicates if the GET_MEM or GET_MEM_WAIT memory request has been fully satisfied. It has the following values: 0 -- Request fully satisfied. 1 -- Request not fully satisfied. 3 -- Request can never be fully satisfied (not enough memory available). 4 -- Unsolicited. **Value_32:** Contains the address of the header of the first buffer in the list of buffer lists. If no buffers are available this field contains the value $FFFFFFFF_{16}$.

DATA_AVAIL

Command Value: 0003_{16}. Transfers a buffer list containing data.
Value_16: Unused.
Value_32: Contains the address of the first buffer in the buffer list.

CLOSE_REQ

Command Value: $000C_{16}$. This packet command is sent by the Client device in the Channel Termination protocol. The Client device sending the CLOSE_REQ command packet must send no further command packets to the receiver via that channel. See Channel Termination Protocol.
Value_16: Unused
Value_32: Unused

CLOSE

Command Value: $000D_{16}$. This packet command is sent by a Server device in the Channel Termination protocol to respond to a CLOSE_REQ command packet from a Client device. The Server device must send no further command packets to the Client device on that channel. See Channel Termination Protocol.
Value_16: Unused
Value_32: Unused

CLOSE_RET

Command Value: $000E_{16}$. This packet command is sent by a Client device in the Channel Termination protocol to return all outstanding Buffer resources to the Server device. See Channel Termination Protocol.
Value_16: Unused
Value_32: Contains the address of the first buffer in the list of buffer lists being returned. If no buffers need to be returned, the Value_32 field is set to $FFFFFFFF_{16}$.

UNKNOWN_COMMAND

Command Value: $000F_{16}$. Sent when a SHARED MEMORY device has received a packet command which it cannot interpret.
Value_16: Contains the command value that could not be interpreted.
Value_32: Unused

TEST

Command Value: 0012_{16}. This packet command is used in the Channel Test protocol. The sending device uses this command to check if the channel is functioning properly. The Value_32 and Value_16 fields together contain a 48-bit unsigned integer to uniquely mark the TEST command. The device receiving the TEST command must immediately respond with the TEST_ACK command.
Value_16: Contains part of the 48-bit sequence number.
Value_32: Contains part of the 48-bit sequence number.

TEST_ACK

Command Value: 0013_{16}. This packet command is used in the Channel Test protocol. The sending device uses this command to respond to the TEST command.
Value_16: Contains the same value received in the TEST command.
Value_32: Contains the same value received in the TEST command.

APID_QUERY

Command Value: 0014_{16}. This packet command is used in the Access Point Identifier resolution protocol. The sending device uses this command to inquire if the receiving device supports a particular Access Point Identifier.
Value_16: Unused.
Value_32: Contains the Access Point Identifier.

APID_SUPPORTED **Command Value: 0015_{16}.** This packet command is used in the Access Point Identifier resolution protocol. The sending device uses this command to respond to the APID_QUERY command when it does support the Access Point Identifier in question.
Value_16: Unused
Value_32: Contains the Access Point Identifier supplied in the APID_QUERY command.

APID_NOT_SUPPORTED **Command Value: 0016_{16}.** This packet command is used in the Access Point Identifier resolution protocol. The sending device uses this command to respond to the APID_QUERY command when it does NOT support the Access Point Identifier in question.
Value_16: Unused
Value_32: Contains the Access Point Identifier supplied in the APID_QUERY command.

UNKNOWN_APID **Command Value: 0017_{16}.** This packet command is sent when a device receives one or more buffers with an Access Point Identifier which it cannot resolve.
Value_16: Unused.
Value_32: Contains the Access Point Identifier in error.

INVALID_ADDR **Command Value: 0018_{16}.** The sending device uses this command to indicate that it has received an invalid address from the sending device.
Value_16: Contains the type of the failed address. It has one of the following values:
1 -- Buffer List address
2 -- Next Buffer address
4 -- Next List address
8 -- Buffer Data Area address.
Value_32: Contains either the Buffer List address (type 1 above) or the address of the Buffer Header containing the invalid address (types 2 & 4 above).

ADDR_RANGE **Command Value: 0019_{16}.** A Server device sends this packet command to determine the address range that a Client device can access.
Value_16: Unused.
Value_32: Unused.

ADDR_RANGE_ACK **Command Value: $001A_{16}$.** A Client device sends this packet command in response to an ADDR_RANGE packet command.
Value_16: Contains the most significant 16 bits of the lowest VMEbus address accessible by the Client device.
Value_32: Contains the most significant 16 bits of the first VMEbus address after the highest VMEbus address accessible by the Client device.

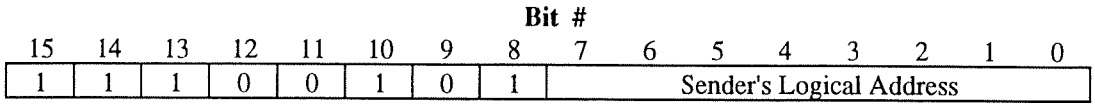
UNEXPECTED_PACKET

Command Value: $001B_{16}$. This packet command is sent when a device receives a GET_MEM or GET_MEM_WAIT packet from a device to which the receiver already has a pending GET_MEM_WAIT packet.
Value_16: Contains the value of the command sent in error (GET_MEM or GET_MEM_WAIT).
Value_32: Unused.

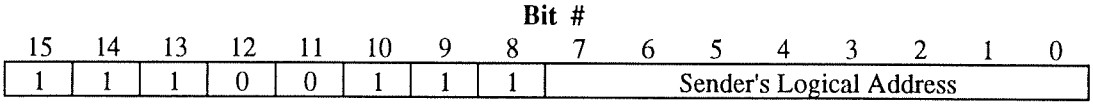
H. PROTOCOL EVENTS

RULE H.1.1:
SHARED MEMORY devices **SHALL** recognize the following events:

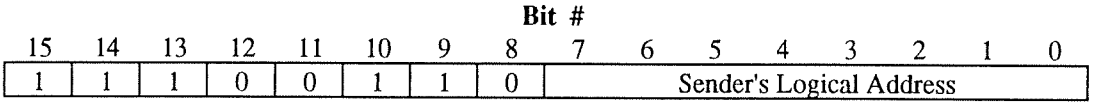
Channel Acknowledge: This event is an acknowledgement that the Channel Setup Data Structure has been read and the SHARED MEMORY channel can now be used.



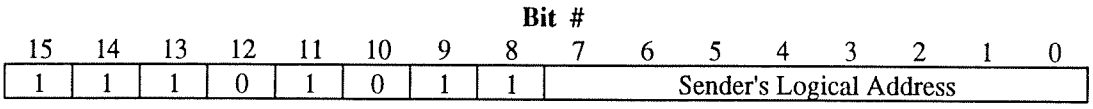
Channel Initiate A24: This event is sent by an A24 SHARED MEMORY Server device to a SHARED MEMORY Client device. It indicates that a Channel Setup Data Structure is available.



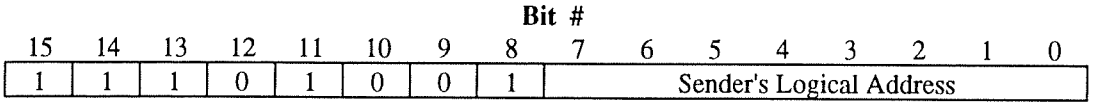
Channel Initiate A32: This event is sent by an A32 SHARED MEMORY Server device to a SHARED MEMORY Client device. It indicates that a Channel Setup Data Structure is available.



Channel Request Asynchronous A24: This event is sent by a SHARED MEMORY Client to request that the event recipient initiate a SHARED MEMORY channel.



Channel Request Asynchronous A32: This event is sent by a SHARED MEMORY Client to request that the event recipient initiate a SHARED MEMORY channel.



Channel Request Synchronous A24: This event is sent by a SHARED MEMORY Client to request that the event recipient initiate a SHARED MEMORY channel.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	1	0	0	Sender's Logical Address							

Channel Request Synchronous A32: This event is sent by a SHARED MEMORY Client to request that the event recipient initiate a SHARED MEMORY channel.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	0	1	0	Sender's Logical Address							

Channel Setup Error: This event is sent by a device when an error has been encountered during the establishment of a SHARED MEMORY channel.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	0	0	0	Sender's Logical Address							

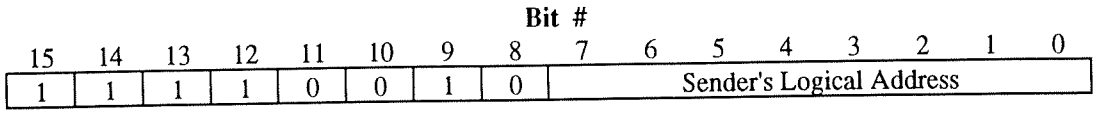
Channel Abort: This event indicates that the channels shared between the event receiver and sender should be aborted.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	0	0	0	0	Sender's Logical Address							

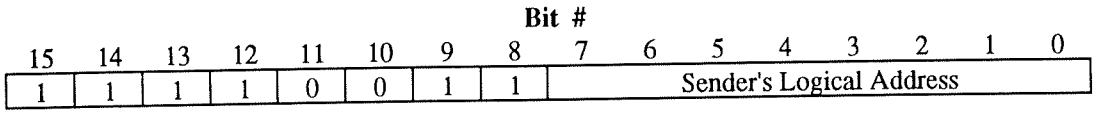
Channel Abort Ack: This event acknowledges that the channels shared between the event sender and receiver have been aborted.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	0	0	0	1	Sender's Logical Address							

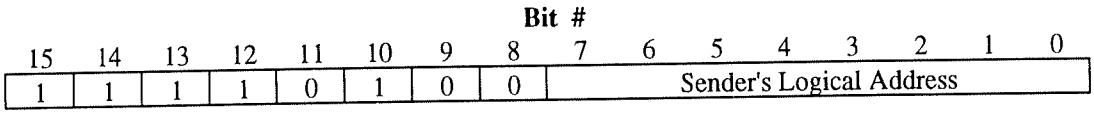
Queue Not Empty Synchronous: This event indicates that the Synchronous packet queue has just changed from being empty to not empty.



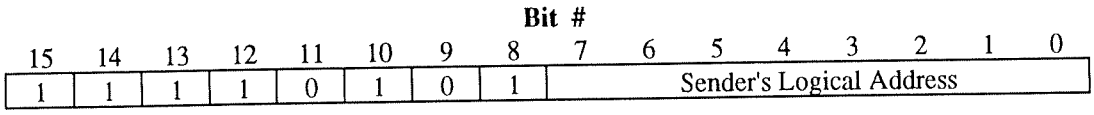
Queue Not Empty Asynchronous: This event indicates that the Asynchronous packet queue has just changed from being empty to not empty.



Queue Not Full Synchronous: This event indicates that the Synchronous packet queue has just changed from being full to not full.



Queue Not Full Asynchronous: This event indicates that the Asynchronous packet queue has just changed from being full to not full.



APPENDIX I. PROTOCOL STATE TABLES

This section contains state tables that model the external behavior of a SHARED MEMORY Communication Protocol implementation. They describe the operation of a single channel, independent of any other channels supported. The three tables model the implementation of a Client device, a Server device and a device that has both Client and Server capability, respectively. A device that is both a Client and a Server behaves differently during connection establishment only. Once the connection is established, the device becomes either a Server or a Client, and its implementation should then follow the model described in the relevant state table.

External state events are generated in one of two ways. Local events are generated by Higher Layer Entities in the local device. These events are identified in the state tables by the word "(local)" next to the event name. All other events are generated by the Shared Memory Communication Protocol executing on other devices in the system. Local events may generate a local error back to the Higher Layer Entity, as specified in the state tables. Though not within the scope of these state tables, note that other activities within the protocol may also cause local errors to be sent to the Higher layer Entity.

For each event, the tables describe the device actions, the messages to be output to the remote device and any subsequent state changes. Output and Next State may vary depending on whether the Action taken succeeds or fails. A blank entry in the Action column indicates that no action other than outputting a message and/or a state transition is required. Events not included in the state tables are to be ignored when received.

The state tables include the following abbreviations and symbols:

Chan	Channel number (the logical address of the remote device)
Copy_CSDS	Copy the CSDS structure from the Server device. A number of conditions (described in the Channel Establishment section) may cause this operation to fail.
Create_CSDS+Queues	Allocate and initialize the CSDS and the queues, if necessary. This operation fails if there are no free CSDS structures or queues available.
Free_CSDS	Mark the CSDS as free.
Free_Queues+RestoreB	Deallocate the queues and free all buffers used by that connection.
Ignore	The received event is ignored. No action required.
LocalError	A local error is returned to the calling Higher Layer Entity.
MyLadd	The logical address of this device.
ProcessInputQueue	Process the input queue.
ProcessOutputQueue	Process the output queue.
Server+Client	A device that has both Server and Client capability.
TryClientConnecting	Attempt to connect as a Client device. May end up switching to a Server role as a result of the response from the remote device.
TryServerConnecting	Attempt to connect as a Server device. May end up switching to a Client role as a result of the response from the remote device.

State	Event	Action	Output (Success)	Output (Failure)	Next State (Success)	Next State (Failure)
Idle	Connect (local)	Copy_CSDS	ChannelRequest	ChannelSetupError	Connecting	Idle
	ChannelAcknowledge		ChannelAbort		Idle	
	ChannelInitiate		ChannelAcknowledge		Data	
	ChannelRequest		ChannelSetupError		Idle	
	ChannelAbort		ChannelAbortAck		Idle	
	QueueNotEmpty		ChannelAbort		Idle	
	QueueNotFull		ChannelAbort		Idle	
Connecting	Connect (local)	LocalError	ChannelAbort	ChannelSetupError	Idle	Idle
	Abort (local)		ChannelAbort		Idle	
	ChannelAcknowledge	Copy_CSDS	ChannelAcknowledge		Data	
	ChannelRequest					
	If Chan>MyLadd		ChannelSetupError		Idle	
	If Chan<MyLadd	Ignore				
	ChannelSetupError				Idle	
	Timeout		ChannelAbortAck		Idle	
	ChannelAbort		ChannelAbort		Idle	
	QueueNotEmpty		ChannelAbort		Idle	
	QueueNotFull				Idle	
Data	Connect (local)	LocalError	ChannelAbort		Idle	
	Abort (local)		CLOSE_REQ (pkt)		Closing	
	Close (local)		ChannelAbort		Idle	
	ChannelAcknowledge		ChannelAbort		Idle	
	ChannelRequest		ChannelAbort		Idle	
	ChannelSetupError		ChannelAbortAck		Idle	
	ChannelAbort				Data	
	QueueNotEmpty	ProcessInputQueue			Data	
	QueueNotFull	ProcessOutputQueue			Idle	
	CLOSE (pkt)		CLOSE_RET (pkt)			
Closing	Connect (local)	LocalError	ChannelAbort		Idle	
	Abort (local)		ChannelAbort		Idle	
	ChannelAcknowledge		ChannelAbort		Idle	
	ChannelRequest		ChannelAbort		Idle	
	ChannelSetupError		ChannelAbortAck		Idle	
	ChannelAbort				Closing	
	QueueNotEmpty	ProcessInputQueue			Closing	
	QueueNotFull	ProcessOutputQueue			Idle	
	CLOSE (pkt)		CLOSE_RET (pkt)			

TABLE I.1 Client State Table

State	Event	Action	Output (Success)	Output (Failure)	Next State (Success)	Next State (Failure)
Idle	Connect (local)	Create_CSDS+Queues	ChannelInitiate		Connecting	Idle
	ChannelAcknowledge		ChannelAbort		Idle	
	ChannelInitiate		ChannelSetupError		Idle	
	ChannelRequest	Create_CSDS+Queues	ChannelInitiate	ChannelSetupError	Connecting	Idle
	QueueNotEmpty		ChannelAbort		Idle	
	QueueNotFull		ChannelAbort		Idle	
Connecting	Connect (local)	LocalError			Aborting	
	Abort (local)		ChannelAbort		Data	
	ChannelAcknowledge	Free_CSDS				
	ChannelInitiate				Idle	
	If Chan>MyLadd	Free_CSDS	ChannelSetupError			
	If Chan<MyLadd	Ignore			Idle	
	ChannelSetupError	Free_CSDS			Aborting	
	Timeout		ChannelAbort		Idle	
	ChannelAbort	Free_CSDS			Idle	
	ChannelAbortAck	Free-CSDS			Aborting	
	QueueNotEmpty		ChannelAbort		Aborting	
	QueueNotFull		ChannelAbort			
Data	Connect (local)	LocalError			Aborting	
	Abort (local)		ChannelAbort		Closing	
	Close (local)		CLOSE (pkt)		Aborting	
	ChannelAcknowledge		ChannelAbort		Aborting	
	ChannelInitiate		ChannelAbort		Aborting	
	ChannelRequest		ChannelAbort		Aborting	
	ChannelSetupError		ChannelAbort		Aborting	
	ChannelAbort	Free_Queues+RestoreB			Idle	
	ChannelAbortAck	Free_Queues+RestoreB			Idle	
	QueueNotEmpty	ProcessInputQueue			Data	
	QueueNotFull	ProcessOutputQueue			Data	
	CLOSE_REQ (pkt)		CLOSE (pkt)		Closing	
Closing	Connect (local)	LocalError			Aborting	
	Abort (local)		ChannelAbort		Aborting	
	ChannelAcknowledge		ChannelAbort		Aborting	
	ChannelInitiate		ChannelAbort		Aborting	
	ChannelRequest		ChannelAbort		Aborting	
	ChannelSetupError		ChannelAbort		Aborting	
	ChannelAbort	Free_Queues+RestoreB			Idle	
	ChannelAbortAck	Free_Queues+RestoreB			Idle	
	QueueNotEmpty	ProcessInputQueue			Closing	
	QueueNotFull	ProcessOutputQueue			Closing	
	CLOSE_RFT (pkt)	Free_Queues+RestoreB			Idle	
Aborting	Connect (local)	LocalError				
	ChannelAbort	FreeQueues+RestoreB			Idle	
	ChannelAbortAck	FreeQueues+RestoreB			Idle	

TABLE I.2 Server State Table

State	Event	Action	Output (Success)	Output (Failure)	Next State (Success)	Next State (Failure)
Idle (Server+Client)	Connect (local)					
	As Client		ChannelRequest		TryClientConnecting	
	As Server	Create_CSDS+Queues	ChannelInitiate		TryServerConnecting	Idle (Server+Client)
	Abort (local)	Ignore				
	ChannelAcknowledge		ChannelAbort		Idle (Server+Client)	
	ChannelInitiate	Copy_CSDS	ChannelAcknowledge	ChannelSetupError	Data (Client)	Idle (Server+Client)
	ChannelRequest	Create_CSDS+Queues	ChannelInitiate	ChannelSetupError	TryServerConnecting	Idle (Server+Client)
	ChannelSetupError	Ignore				
	ChannelAbort		ChannelAbortAck		Idle (Server+Client)	
	ChannelAbortAck	Ignore				
	QueueNotEmpty		ChannelAbort		Idle (Server+Client)	
	QueueNotFull		ChannelAbort		Idle (Server+Client)	
TryServerConnecting	Connect (local)	LocalError				
	Abort (local)		ChannelAbort		Aborting (Server)	
	ChannelAcknowledge	Free_CSDS			Data (Server)	
	ChannelInitiate					
	If Chan>MyLadd	Free_CSDS,Copy_CSDS	ChannelAcknowledge	ChannelSetupError	Data (Client)	Idle (Server+Client)
	If Chan<MyLadd	Ignore				
	ChannelRequest	Ignore				
	ChannelSetupError	Free_CSDS			Idle (Server+Client)	
	Timeout		ChannelAbort		Aborting (Server)	
	ChannelAbort	Free_CSDS			Idle (Server+Client)	
	ChannelAbortAck	Free_CSDS			Idle (Server+Client)	
	QueueNotEmpty		ChannelAbort		Aborting (Server)	
	QueueNotFull		ChannelAbort		Aborting (Server)	
TryClientConnecting	Connect (local)	LocalError				
	Abort (local)		ChannelAbort		Idle (Server+Client)	
	ChannelAcknowledge		ChannelAbort		Idle (Server+Client)	
	ChannelInitiate	Copy_CSDS	ChannelAcknowledge	ChannelSetupError	Data (Client)	Idle
	ChannelRequest					
	If Chan>MyLadd	Create_CSDS+Queues	ChannelInitiate	ChannelSetupError	TryServerConnecting	Idle (Server+Client)
	If Chan<MyLadd	Ignore				
	ChannelSetupError				Idle (Server+Client)	
	Timeout				Idle (Server+Client)	
	ChannelAbort		ChannelAbortAck		Idle (Server+Client)	
	QueueNotEmpty		ChannelAbort		Idle (Server+Client)	
	QueueNotFull		ChannelAbort		Idle (Server+Client)	

TABLE I.3 Server+Client State Table

APPENDIX II. REVISION 1.2 COMPATIBILITY

Compatibility with the VXIbus System Specification Revision 1.2 is maintained through the SHARED MEMORY Channel Type parameter. In Revision 1.2, Synchronous and Asynchronous channels were assigned types FF_{16} and FE_{16}, respectively. This is in contrast with the current channel types 80_{16} and 81_{16}. During channel establishment, if the Client does not support the channel type specified by the Server, the Client simply sends the **Channel Setup Error** event to the Server. This immediately terminates the connection attempt.

This revision of the SHARED MEMORY Protocol differs from Revision 1.2 in the following areas:

- The size of SHARED MEMORY buffers is no longer fixed for a particular channel. A Buffer Size field has been added to the buffer header allowing buffers to have variable sizes even within a single buffer list. The Buffer Size field in the Channel Setup Data Structure (CSDS) has been removed.

- The SHARED MEMORY Buffer Header has been separated from the Buffer Data Area, and the address of the Buffer Data Area has been added to the Buffer Header.

- A number of the fields in the buffer header have been modified.

- The two Sub_Address fields in the buffer header have been replaced with a single Access Point Identifier field.

- This revision of the SHARED MEMORY Protocol allows only one Synchronous channel and one Asynchronous channel to be established between any two SHARED MEMORY devices. Multiple higher layer connections are multiplexed over these two channels.

- The GET_MEM and CLOSE_ACK packet commands now have a value defined for their Value_16 fields, and the GET_MEM_WAIT packet command has been added.

- The RET_MEM packet command has been replaced by the MEM_AVAIL packet command. The RET_BUFS packet command has been removed.

- The Channel Clear and Channel Reset protocols have been removed and the Channel Abort and Channel Test protocols have been added.

- In Revision 1.2 a number of commands and events were included to allow a VXIbus commander to control the SHARED MEMORY connections of its servants. This was found to be a higher level function and was therefore removed from this revision of the Protocol.

US Sales and Service Offices

ALABAMA

Huntsville
4970 Corporate Drive,
Suite 100
Huntsville 35805
Phone: (205) 830-9212
FAX: (205) 837-2666

ALASKA

Served by Frigid North
Company (Representative),
Anchorage, Alaska,
(907) 561-4633

ARIZONA

Phoenix
3015 S. 48th Street,
Suite 100
Tempe 85282
Phone: (602) 438-1011
FAX: (602) 437-9210
Mailing Address:
P.O. Box 29540
Phoenix 85038

ARKANSAS

Served by EnTest, Inc.
(Representative), Dallas,
Texas, (800) 955-0077

CALIFORNIA

Walnut Creek
1340 Treat Blvd., Suite 660
Walnut Creek 94596
Phone: (510) 932-4949
Fax: (510) 932-8596

Irvine
17052 Jamboree Blvd.
Irvine 92714
Phone: (714) 660-8080
TELEFAX: (GP1)
(714) 660-8080 X311
FAX: (714) 474-1490
Mailing Address:
P.O. Box 19523
Irvine 92713

Los Angeles
21300 Erwin Street
Woodland Hills 91367
Phone: (818) 999-1711
FAX: (818) 340-9840
Mailing Address:
P.O. Box 8500
Woodland Hills 91365

Manhattan Beach
3601 Aviation Blvd.,
Suite 1200
Manhattan Beach 90266
Phone: (213) 643-6605

San Diego
8810 Rehco Rd.,
Bldg. D, Suite A
San Diego 92121
Phone: (619) 558-3221
FAX: (619) 558-3232

Santa Clara
3003 Bunker Hill Lane
Santa Clara 95052-9937
Phone: (408) 496-0800
TELEFAX: (GP1)
(408) 496-0800 X137
FAX: (408) 988-6051

COLORADO

Denver
393 Inverness Drive South
Englewood 80112
Phone: (303) 799-1000
FAX: (303) 790-4754
Mailing Address:
P.O. Box 6979
Englewood 80155-6979

CONNECTICUT

Served by the Boston,
Massachusetts field office

DELAWARE

Served by the Philadelphia,
Pennsylvania field office

DISTRICT OF COLUMBIA

See MARYLAND

FLORIDA

Orlando
12051 Corporate Blvd.
Orlando 32817
Phone: (407) 249-1600
From the Cape Canaveral
Area: (407) 636-0343
FAX: (407) 277-4723

Pensacola
4700 Bayou Blvd., Bldg. 1
Pensacola 32503
Phone: (904) 476-1897
FAX: (904) 476-0720

GEORGIA

Atlanta
Technology Park/Atlanta
650 Engineering Drive
Norcross 30092
Phone: (404) 449-4770
FAX: (404) 449-0358
Mailing Address:
P.O. Box 6500
Norcross 30091

HAWAII

Honolulu
ComTest Technologies, Inc.
3049 Ualena St., Suite 1005
Honolulu, Hawaii 96819
Phone: (808) 831-0020
FAX: (808) 831-0017

IDAHO

Served by Seltech, Inc.
(Representative), Bellevue,
Washington
(206) 746-7970

ILLINOIS

Chicago
5350 Keystone Court
Rolling Meadows 60008
Phone: (708) 259-7580
FAX: (312) 259-8388

INDIANA

Indianapolis
8751 Wesleyan Road
Indianapolis 46268
Phone: (317) 872-3708
FAX: (317) 875-0661

IOWA

Served by the Chicago,
Illinois field office
Phone: (708) 259-7580 or
Dallas, Texas field office
Phone: (800) 852-4043

KANSAS

Kansas City
Served by the Dallas, Texas
field office
Phone: (800) 852-4043

KENTUCKY

Served by the Dayton, Ohio;
St. Louis, Missouri; and
Indianapolis, Indiana field
offices

LOUISIANA

Served by EnTest, Inc.
(Representative), Dallas,
Texas, (800) 955-0077

MAINE

Served by the Boston,
Massachusetts field office

MARYLAND

DC
700 Professional Drive
Gaithersburg 20879-3499
Phone: (301) 948-7151
FAX: (301) 921-0461
Mailing Address:
P.O. Box 6026
Gaithersburg 20884-0986

MASSACHUSETTS

Boston
482 Bedford Street
Lexington 02173
Phone: (617) 861-6800
FAX: (617) 863-5994

MICHIGAN

Detroit
37890 Interchange Drive
Farmington Hills 48331
Phone: (313) 478-5200
FAX: (313) 478-1521

MINNESOTA

St. Paul
2685 Long Lake Road
Building A
Roseville 55113
Phone: (612) 635-0520
FAX: (612) 635-0716

MISSISSIPPI

Served by EnTest, Inc.
(Representative), Dallas,
Texas, (800) 955-0077

MISSOURI

St. Louis
1854 Lackland Hill Parkway
St. Louis 63146
Phone: (314) 569-1060
FAX: (314) 569-1061

MONTANA

Served by Seltech, Inc.
(Representative), Bellevue,
Washington
(206) 746-7970

NEBRASKA

Served by the Tektronix
T&M National Marketing
Center, Beaverton, Oregon
(800) 426-2200

NEVADA

Served by the Walnut Creek,
California and Albuquerque,
New Mexico field offices

NEW HAMPSHIRE

Served by the Boston,
Massachusetts field office

NEW JERSEY

Woodbridge
40 Gill Lane
Woodbridge 07095
Phone: (201) 636-8616
FAX: (201) 636-7234

NEW MEXICO

Albuquerque
6100 Uptown Blvd.
Suite 640
Albuquerque 87110
Phone: (505) 884-2828
FAX: (505) 884-8558

NEW YORK

Albany
26 Computer Drive West
Albany 12205
Phone: (518) 458-7291
FAX: (518) 458-7361

Long Island
225 Wireless Blvd.
Hauppauge 11788
Phone: (516) 231-9400
From NYC: (718) 895-8010
FAX: (516) 231-9624

Rochester
1160 Pittsford-Victor Road,
Suite C
Pittsford 14534
Phone: (716) 383-0070
FAX: (716) 383-0022

NORTH CAROLINA

Raleigh
2950 Gateway Centre Blvd
Morrisville 27560
Phone: (919) 481-3800
FAX: (919) 469-8823

NORTH DAKOTA

Served by the Tektronix
T&M National Marketing
Center, Beaverton, Oregon
(800) 426-2200

OHIO

Dayton
8571 Gander Creek Drive
Miamisburg 45342
Phone: (513) 438-1166
FAX: (513) 438-1106

OKLAHOMA

Oklahoma City
Served by the Dallas, Texas
field office
Phone: (800) 852-4043

OREGON

Portland
Mailing Address:
Portland Sales M/S 02-PTLD
P.O. Box 500
Beaverton 97077
Phone: (503) 627-7999
FAX: (503) 627-7995
Shipping Address
Portland Sales M/S:
02-PTLD
Regional Distribution Center
Beaverton 97077

Factory Service Center:
Tektronix Industrial Park
Beaverton 97077
Phone: (503) 627-6630
1-800-835-9433
TLX: 192825 TEKTRONIX

PENNSYLVANIA

Philadelphia
450 Sentry Parkway
Blue Bell 19422
Phone: (215) 825-6400
FAX: (215) 825-8839

Pittsburgh
1051 Brinton Road,
Suite 300
Pittsburgh 15221
Phone: (412) 244-9800
FAX: (412) 244-1948

RHODE ISLAND

Served by the Boston,
Massachusetts field office

SOUTH CAROLINA

Served by the Atlanta,
Georgia and Raleigh, North
Carolina field offices

SOUTH DAKOTA

Served by the Tektronix
T&M National Marketing
Center, Beaverton, Oregon
(800) 426-2200

TENNESSEE

Served by the Atlanta,
Georgia field office

TEXAS

Austin
Served by the Houston and
Dallas, Texas field offices

Dallas
1551 Corporate Drive
Irving 75038
Phone: (214) 550-0525
Metro: (214) 751-0470
Sales FAX:
(214) 550-8226
Service FAX:
(214) 550-7363
Mailing Address:
P.O. Box 165027
Irving 75016

El Paso
10432 Brian Mooney,
Suite 101
El Paso 79935
Sales:
(915) 593-8090
Service:
(915) 593-8095

Houston
One West Chase Center
10777 Westheimer
Suite 675
Houston 77042
Phone: 1-800-847-0407
(713) 975-8670
FAX: (713) 784-4440

Kelly
1831 S. General
McMullen Dr., Suite 101
San Antonio 78226
Phone: (512) 432-1341
FAX: (512) 432-1859

UTAH

Served by the Seattle,
Washington field office

VERMONT

Served by the Albany, New
York field office

VIRGINIA

Crystal City
Crystal Square 2
1725 Jefferson Davis Hwy.,
Suite 406
Arlington 22202
Phone: (703) 486-1300
FAX: (703) 486-1305

Hampton
525 Butler Farm Road,
Suite 102
Hampton 23666
Phone: (804) 865-1588
FAX: (804) 865-7892

WASHINGTON

Seattle
11814 115th Avenue NE
Kirkland 98034
Phone: (206) 821-9100
FAX: (206) 821-3607

WEST VIRGINIA

Served by the Pittsburgh,
Pennsylvania
field office

WISCONSIN

Served by the Chicago,
Illinois, and St. Paul,
Minnesota field offices

WYOMING

Served by the Tektronix
T&M National Marketing
Center, Beaverton, Oregon
(800) 426-2200

CORPORATE OFFICE

Tektronix, Inc.
26600 SW Parkway
PO Box 1000
Wilsonville, Oregon
97070-1000
(503) 682-3411
(800) 426-2200

Worldwide Sales and Service Offices

For customers in areas not listed, see below for your nearest office.

Customers in Eastern Europe, Near- and Middle East contact: Tektronix Ges.m.b.H., Austria

Customers in Benin, Burkina Faso, Cameroon, Central African Republic, Chad, Congo, Djibouti, Equatorial Guinea, Gabon, Guinea, Ivory Coast, Malagasy, Mali, Mauritius, Niger, Senegal, Togo, Zaire contact: Tektronix S.A., France

Customers in Andorra, Angola, Azores, Gibraltar contact: Tektronix Española, S.A., Spain

Customers in unlisted African, South American, Pacific Island, or Asian locations contact:
Tektronix , Inc., U.S.A.
Export Sales
P.O. Box 500 M/S 50-439
Beaverton, OR 97077 USA
Phone:
1-800-835-9433 x1916
Telex: 192825 TEKTRONIX
Fax: (503) 627-6905

ALGERIA

(Product Information & Service)

TEKSYSTEM
36 Place Ben Omar
KOUBA
ALGERIA
Phone: 213 (*) 268.05.28
Fax: same

(Sales Office)

Tektronix, S.A.
Algeria Sales
ZA de Courtaboeuf, Av du Canada, BP 13
91941 Les Ulis Cédex
France
Phone: 33 (1) 69 86 81 81
FAX: 33 (1) 69 07 09 37
Telex: (842) 604332 TEKOR A

ARGENTINA

Coasin S.A.
Buenos Aires
Virrey del Pino 4071
1430 Buenos Aires
Phone: 54 (1) 552-5248/-3485/-3185
FAX:
54 (1) 11-1427 (International)
551-9361 (Local)
Telex:
(390) 17014 COASIN AR

Cordoba
25 de Mayo No. 1930
5000 Cordoba
Phone: 54 (51) 3037

AUSTRALIA

Tektronix Australia Pty. Limited

Sydney Office
80 Waterloo Road
North Ryde, N.S.W. 2113
Phone: 61 (2) 888-7066
FAX: 61 (2) 888-0125

Adelaide
128 Gilles Street
Adelaide, South Australia 5000
Phone: 61 (8) 223-2811
FAX: 61 (8) 223-7890

Brisbane
737 Logan Road
Greenslopes Brisbane
Queensland 4120
Phone: 61 (7) 394-1155
Fax: 61 (7) 397-8679

Canberra
Unit 14
Trevor Pearcy House
Fern Hill
Bruce, Austr.
Capital Terr. 2617
Phone: 61 (6) 251-6111
FAX: 61 (6) 251-4509

Perth
The Garden Office Park
345 Harborne Street
Herdsman,
West Australia 6017
Phone: 61 (9) 242-4688
Fax: 61 (9) 242-4707

Victoria Branch Office
259-263 Whitehorse Rd.
Balwyn, Victoria 3103
Phone: 61 (3) 836-3355
FAX: 61 (3) 836-6592

AUSTRIA

Tektronix Ges.m.b.H.
Doerenkampgasse 7
A-1100 Vienna, Austria
Phone: 43(222) 68-66-02-0
FAX: 43(222) 68-66-00
Telex: (847) 111481 TEK A

BAHRAIN

International Agencies Co Ltd.
(Intercol)
P.O. Box 584
Manama 301
Phone: 973(*)727 177
FAX: 973(*) 727 509
Telex:
(490) 8661 INTCOL BN

BANGLADESH

A. Q. Chowdhury & Co.
KJH Mansion
Spectachrome Floor
83 Laboratory Road
Dhaka – 1205
Phone: 880 (2) 861910, 861911, 509051
FAX: 880 (2) 863475
Telex: (950) 671259 AQC BJ

BELGIUM

Tektronix nv/sa
Brussels Office
Bedrijfspark Keiberg
Excelsiorlaan, 3
1930 Zaventem
Phone: 32(2) 725-9610
FAX: 32(2) 725-9953
Telex:
(846) 26713 TEKBEL B

BOLIVIA

Coasin Bolivia S.R.L.
Calle Gabriel Gozalvez 221
PO Box 7295
La Paz
Phone: 591 (2) 340962, 363365
FAX: 591 (2) 35 9268
Telex:
(336) 3233 COALAP BV

BRAZIL

Tektronix Industria e Comercio Ltda.
São Paulo Office
Av. Vereador José Diniz, 3530
CEP 04604-006-Campo Belo
São Paulo, SP
Phone: 55 (11) 543.1911
FAX: 55 (11) 542.0696
Telex:
(391) 1154068 TEKX BR

Rio de Janeiro Office
Av. Marechal Câmara, 160-cj.605
CEP 20020 – 080 Centro
Rio de Janeiro, RJ
Phone: 55 (21) 240.7370
FAX: 55 (21) 240.7793
Telex:
(391) 2130120 TEKX BR

BULGARIA

New-Tek Ltd.
Technical University
Room 2325
Darvenitsa Bloc 2
BG-1156 Sofia
Phone: (2) 66 53 46
FAX: (2) 68 71 10
Telex:
(860) 22958 SVCE BG

CANADA

Tektronix Canada Inc.
Barrie
50 Alliance Blvd.
Mail: P.O.Box 6500,
Barrie, Ontario L4M 4V3
Phone: (705) 737-2700
FAX: (705) 737-5588

Calgary
Suite 106
3016, 19th Street NE
Calgary, Alberta T2E 6Y9
Phone: (403) 250-7583
FAX: (403) 291-9840

Montreal
900 Selkirk Street
Pointe Claire,
Quebec H9R 3S3
Phone: (514) 697-5340
FAX: (514) 697-5205

Ottawa
1642 Woodward Drive
Ottawa, Ontario K2C 3R8
Phone: (613) 225-2850
FAX: (613) 225-9719

Toronto
1200 Courtney Park Dr.
Mississauga, Ontario
L5T 1S2
Phone: (416) 670-3423
FAX: (416) 670-4229

Vancouver
4175 Dawson Street
Burnaby, B.C. V5C 4B3
Phone: (604) 291-1222
FAX: (604) 291-1624

CHILE

(Test & Measurement and Television only)

Equipos Industriales S.A.C.I.
Moneda 812, Of. 905
Santiago
Phone: 56 (2) 6990506
FAX: 56 (2) 633 4509
Telex: 56 (2) 633 4539
(359) 340987 FLOBRA CK
(359) 341177 FLOBRA CK

(Computer Graphics and Printers only)

Ingenac S.A.
Av. Pedro de Valdivia 800
Providencia, Santiago
Phone: 56 (2) 233-9935, 233-9941, 233-9126, 274-0973
Fax: 56 (2) 234-1107

Tektronix Sales Office
Quebec 496
Providencia, Santiago
Phone: 56 (2) 204-9305, 204-9144
FAX: 56 (2) 204-7476

CHINA, PEOPLE'S REPUBLIC OF

Tektronix China, Ltd.
Beijing Office
Room K&M
7th Floor Office Building
New Century Hotel
6 Shoudutiyuguan
South Road
Beijing, P.R.C.
Phone: 86 (1) 832 7657, 849 2340
FAX: 86 (1) 832 7502
Telex:
(716) 221095 NCH CN

Shanghai Office
Room 614/616,
Shanghai Convention Centre
500 Zhao Jia Bang Road
Shanghai P.R.C.
Phone: 86 (21) 471 0281, 471 6060, 471 2614
FAX: 86 (21) 471 0282

Hong Kong
22nd Floor,
Dah Sing Financial Ctr
108 Gloucester Road
Wanchai, Hong Kong
Phone: 852 (*) 598 6188
FAX: 852 (*) 598 6260
Telex:
(780) 77708 DCBIA HX

U.S. Office
P.O. Box 500, M/S 50-438
Beaverton, Oregon 97077
U.S.A.
Phone: (503) 627-6546
FAX: (503) 627-6905
Telex 192825 TEKTRONIX

CIS

Nivex
1st Brestskaja Street, 29/22
1st Floor
Phone: 57 (1) 263-1220,
P.o. Box 119
SU-125047 Moscow
Phone: 70 (95) 371 45 89

COLOMBIA

Selectronica, Ltda.
Carrera 71 #55-98
Mail: Apartado Aereo 25124
Bogota D.E.
Phone: 57 (1) 263-1220, 295-3215
FAX: 57 (1) 263-2874
Cable: SELECTRONICA

COSTA RICA

Electroimpex, S.A.
200 Metros oeste de
Pops La Sabana
Carretera a Escazu
Mail: Apartado 620-1000
San Jose
Phone: 506 (*) 31-5701, 31-3426
FAX: 506 (*) 31-6531
Telex:
(375) 2645 ELEPEX CR

CROATIA

Tescom
Gunduliceva 41
CRO-41000 Zagreb
Phone: 38 (41) 417 080
FAX: 38 (41) 442 273

CROATIA

Chris Radiovision Ltd.
P.O. Box 1989
Nicosia
Phone: 357 (2) 466121
FAX: 357 (2) 365177
Telex: (826) 605 2395

CZECHOSLOVAKIA

Seminar & Training Center
Bartolomejska 13
CS-11000 Praha 1
Phone: 42 (2) 22 32 63
FAX: 42 (2) 236 13 46

Zenit
Seminar & Training Center
Radlicka 138
CS-15000 Praha 5
Phone: 42 (2) 53 69 21
FAX: 42 (2) 53 62 93
Telex: (849) 121801

DENMARK

Tektronix A/S
Literbuen 7
DK-2740 Skovlunde
Phone: 45 44 53 54 55
FAX: 45 44 53 07 55

ECUADOR

Proteco Coasin Cia Ltda. Quito
12 de Octubre 2449 y Orellana
Casilla 7-03-228A Quito
Phone: 593 (2) 52-6759, 52-9684
FAX: 593 (2) 56-1980
Telex:
(393) 22865 PROTEC ED
22085 ESAND ED

Guayaquil
Miraflores Av. Principal 204
y Calle Segunda
Guayaquil
Phone: 593 (4) 201529

EGYPT

Giza Systems Engineering Co.
2, El Mesaha Sq.
DOKKI, A.R.E.
P.O. Box 1913
CAIRO 11511
Phone: 20 (2) 3490140
FAX: 20 (2) 3609932
Telex: 93644 ALKAN

FINLAND

Tektronix Oy
Larin Kyöstin tie 4,
SF-00650 Helsinki 65
Phone: 358(0) 7282 400
FAX: 358(0) 7520033

FRANCE

Tektronix S.A.
Orsay
ZAC Courtaboeuf,.
4 Av. du Canada B.P.13
91941 Les Ulis Cédex
Phone: 33(1) 69 86 81 81
FAX: 33(1) 69 07 09 37
Telex: (842) 604332 TEKOR A

After Sales Service—France
ZAC. Courtaboeuf,
4 Av. du Canada
B.P.13
91941 Les Ulis Cédex
Phone: 33(1) 69 86 81 81
FAX: 33(1) 69 07 09 37
Telex:
(842) 601012 TEKSAV
Appels interventions après-vente France métropolitaine: numéro vert 05 27 38 04

Aix-en-Provence Region Center
Immeuble Tertia 1-ZI d'Aix
13763 Les-Milles Cédex
Phone: 33 (16) 42 24 45 03
FAX: 33 (16) 42 24 41 20
Telex: (842) 440045

Lyon Region Center
Les Jardins d'Entreprise
213 rue de Gerland, Bât. F1
69007 LYON
Phone: 33 (16) 78 58 15 38
FAX: 33 (16) 78 69 52 55
Telex:
(842) 300150 TEKLYON

Worldwide Sales and Service Offices

Rennes Region Center
ZAC de la Rigourdière
35510 Cesson Sévigné
Phone: 33 (16) 99 83 40 40
FAX: 33 (16) 99 83 36 23
Telex:
(842) 740829 TEKREN

Strasbourg Region Center
Parc d'Activités des Tanneries
2a, Route de la Rivière
67380 Lingolsheim
Adresse postale:
BP 98-
67833 Tanneries Cedex
Phone: 33 (16) 88 76 15 45
FAX: 33 (16) 88 76 09 89
Telex: (842) 890470

Toulouse Region Center
Peri-Sud 2, Z.I. du Palays
13 Rue Villet Bat. 1
31400 Toulouse, France
Phone: 33 (16) 61 80 25 69
FAX: 33 (16) 61 34 84 52
Telex:
(842) 530243 TEKTOUL F

GERMANY, FEDERAL REPUBLIC OF

Tektronix GmbH
Colonia Allee 11
5000 Köln 80
Tel: 49 (221) 9 69 69-0
FAX: 49 (221) 96969-362

Geschäftsstellen:
Ernst-Reuter-Platz 3-5
1000 Berlin 10
Tel: (030) 31 59 05-0
FAX: (030) 3 13 31 32

Albert-Einstein-Ring 9
2000 Hamburg 50
Tel: (040) 8 90 98-0
FAX: (040) 8 90 98-159

Lyoner Straße 14
6000 Frankfurt 71
Tel: (069) 66 90 02-0
FAX: (069) 6 66 40 93

Neureuter Straße 37a-b
7500 Karlsruhe 21
Tel: (0721) 79 080
FAX: (0721) 7908-300

Ehrenbreitsteiner Straße 28
8000 München 50
Tel: (089) 14 85-0
FAX: (089) 1 48 52 50

GREECE

HITEC, S.A.
5, Chimeras Str.
Marousi 15125
Athens
Phone: 30 (1) 68 99 340
FAX: 30 (1) 68 99 342

GUAM

(Color Printers Only)
MiTech International
130 E. Marine Drive,
Suite 103B
Agana, Guam 96910
Phone: (671) 477-5423
FAX: (671) 477-5424

HONG KONG

Tektronix Hong Kong, Ltd.
22nd Floor,
Dah Sing Financial Ctr
108 Gloucester Road
Wanchai, Hong Kong
Phone: 852 (*) 598 6188
FAX: 852 (*) 598 6260
Cable: TEKHKGLTD

HUNGARY

Tektronix Representation Office Hungary
Horvat U. 2-12. IV.EM.39.
H-1027 Budapest
Phone: 852 (*) 598-6188
FAX: 852 (*) 598-6260

INDIA

Hinditron Tektronix Instruments, Ltd.
Bangalore
No. 5, Crescent Rd
High Grounds
Bangalore 560 001
Phone: 91 (812) 265-470,
265-471, 263-474
FAX: 91 (812) 345022
Telex:
(953) 8452151 HIPL IN
Cable: TEKHIND

Bombay
Industry House
23-B, Mahal Industrial Estate
Mahakali Caves Rd
Andheri (East)
Bombay 400 093
Phone: 91 (22) 6364560
FAX: 91 (22) 6364682
Telex:
(953) 01179286 HIPL IN
Cable: MAHALHIND

Calcutta
Castle House, 5th Floor,
5/1-A, Hungerford Street,
Calcutta 700 017
Phone: 91 (33) 476844,
479094
FAX: 91 (33) 751799
Telex:
(953) 0214153 HSPL IN
Cable: FLUKHIND

Lucknow
Mr. S.K. Sangal
15/30 Indiranagar
Lucknow 226016

Madras
No. 22, II, Balaji Avenue
Off Thirumalai Pillai Rd.
Madras 600 017
Phone: 91 (44) 864 699,
861286, 869480
Telex:
(953) 0416188 HCPL IN
Cable: FLUKHIND

New Delhi
201-206 Hemkunt Tower
98, Nehru Place
New Delhi 110 019
Phone: 91 (11) 641-0380,
643-0519, 643-5278,
644-4234
FAX: 91 (11) 644-2728,
644-7032
Telex:
(953) 03161458 HSPL IN
Cable: TEKCENTRE

Pune
Mira Co-op Housing
Society Ltd
Flat 118, 3rd Floor
Building 8
550/1 Salisbury Park
Pune, INDIA 411 001
Phone: 91 (212) 654567

Secunderabad
5th Floor, Emerald House
114 Sarojini Devi Road
Secunderabad 500 003
Phone: 91 (842) 844033,
843753
Telex:
(953) 04256973 HCPL IN
Cable: TEKHIND

INDONESIA

Mecomb Tehnik
Bldg. #304
Cilandak Commercial Estate
Jl Cilandak KKO
PO Box 1 CCE
Jakarta 12560
Phone: 62 (21) 780 8008
FAX: 62 (21) 780-1040

IRAN

Vision International N.V.
Researchpark
Interleuvenlaan 64
B-3001 Leuven
Belgium
Phone: 32 (16) 40 04 16
FAX: 32 (16) 40 03 52
Telex: (846) 26202

IRELAND

Tektronix U.K. Ltd
Deans Grange Estate,
Kill Lane,
Blackrock,
County Dublin,
Ireland
Phone: 353 (01) 2892166,
2892185
FAX: 353 (01) 2892106
Telex: (852) 90478 TEKX EI

ISRAEL

Eastronics Ltd
11 Rozanis Street
Mail:
P.O. Box 39300
Tel Aviv 61392
Phone: 972 (3) 475151
FAX: 972 (3) 475125
Telex:
(922) 33638 RONIX IL

ITALY

Tektronix S.p.A.
Milan
Via Lampedusa 13
20141 Milan
Phone: 39(2) 84441
FAX: 39(2) 8950-0665
Telex:
(843) 315217 TEKMIL I

Rome
Piazza Antonio Baldini 45
00141 Rome
Phone: 39 (6) 827-1041,
827-1039
FAX: 39 (6) 868-91546
Telex:
(843) 616018 TEKROM I

Turin
Via Cardinal M. Fossati 5
10141 Turin
Phone: 39 (11) 385-1143,
385-2163, 385-4086
FAX: 39 (11) 3352783
Telex:
(843) 212543 TEKTOR I

JAPAN

Sony/Tektronix Corporation Headquarters
9-31, Kitashinagawa 5
Shinagawa-ku
Tokyo 141
Mail: P.O. Box 5209
Tokyo Int'l, Tokyo 100-31
Phone: 81(3) 3448-4611
FAX: 81(3) 3444-0318
Telex:
(781) 2422850 STTYO J

Atsugi
Pastoral Bldg
1-8-6 Asahi-cho
Atsugi 243
Phone: 81 (462) 28-6768
FAX: 81 (462) 28-8043

Chiba
Taiso Center Bldg.
2-7-2 Shinchiba
Chiba 260
Phone: 81 (472) 42-1911
FAX: 81 (472) 42-2291

Fukuoka
Nihon Seimei
Hakataekimae Bldg.
2-1 Hakataekimae-3,
Hakata-ku
Fukuoka 812
Phone: 81 (92) 472-2626
FAX: 81 (92) 472-2627

Hiroshima
HIROSHIMA INTES
10-10 Hashimoto-cho
Naka-ku, Hiroshima 730
Phone: 81 (82) 222-7221
FAX: 81 (82) 222-7225

Nagoya
Sanko Building
21-7 Meieki 3-Chome,
Nakamura-ku
Nagoya 450
Phone: 81 (52) 581-3547
FAX: 81 (52) 563-7637

Omiya
Omiya Sawada Building
1-12-5 Sakuragicho
Omiya 331
Phone: 81 (48) 646-0711
FAX: 81 (48) 646-0859

Osaka
Osaka Ohbayashi Building
4-33 Kitahama-Higashi
Chuo-ku, Osaka 540
Phone: 81 (6) 947-0321
FAX: 81 (6) 941-3861

Sendai
Sendai NS Building
2-25 Ichibancho-1
Aoba-ku, Sendai 980
Phone: 81 (22) 267-2181
FAX: 81 (22) 262-3290

Tama
2-45-10 Fujimidai
Kunitachi, Tokyo 186
Phone: 81 (425) 73-2111
FAX: 81 (425) 73-2170

Tsuchiura
Sumitomo Seimei
Tsuchiura Bldg.
16-12 Sakuracho-1
Tsuchiura 300
Phone: 81 (298) 24-2602
FAX: 81 (298) 24-2606

Yokohama
Dai-ichi Yamashitacho Bldg.
74-1 Yamashitacho
Naka-ku, Yokohama 231
Phone: 81 (45) 661-3511
FAX: 81 (45) 661-3519

JORDAN

Consult-Tel
P.O. Box 35055
Amman
Phone: 962 (6) 636668
FAX: 962 (6) 637668
Telex:
(493) 22345 CONSLT JO

KOREA, REPUBLIC OF

Myoung Corporation Seoul Office
Sinsong Bldg., 25-4
Yeoeuido-Dong,
Youngdeungpo-ku
Mail: Yeoeuido
P.O. Box 14 Seoul
Phone:
82 (2) 784-9942
(representative)
FAX: 82 (2) 784-2387
Telex:
(787) 24283 MYOUNG K

Taegu Office
139-5, Shincheon-Dong,
Dong-ku, Taegu City
Phone: 82 (53) 753-8163/4
FAX: 82 (53) 753-8165

Taejon Office
Hanjin Officetel Rm 1202
551-7, Bongmyoung-Dong,
Yusung-ku
Taejon
Phone: 82 (42) 823-9942
FAX: 82 (42) 822-2130

Myoung System, Inc.
Kyoungsin Building,
2nd Floor
1049-4 Sadang-Dong,
Dongjak-ku
Seoul 156-091
Phone: 82 (2)
522-9942 (rep.)
FAX: 82 (2) 522-9943

KUWAIT

Tareq Company Ltd.
P.O. Box 20506
13066, Safat, Kuwait
Phone: 965 (*) 243 6100
FAX: 965 (*) 243 7700
Telex:
(496) 22315 ZUAITER KT

LEBANON

Mideast Data Systems
P.O. Box 7899
Abu Dhabi
United Arab Emirates
Phone: 971 (2) 213 000
FAX: 971 (2) 213 714
Telex:
(893) 24 325 MDSUAE EM

MALAYSIA

Mecomb Malaysia Sdn. Bhd.
Lot 20, Jalan 225
46100 Petaling Jaya
Mail:
P.O. Box 24
46700 Petaling Jaya
Selangor
Phone: 60 (3) 7743422
FAX: 60 (3) 7743414
Telex: (784) 37764 MA

Ipoh Office
Bangunan Tractors
Batu 4, Jalan Lahat
Bukit Merah
30730 Ipoh
Phone: 60 (5) 221-239
Fax: 60 (5) 210-543

Penang Office
No. 2487, Jalan Glugor
11700 Penang
Phone: 60 (4) 373042
Fax: 60(4) 883658

Sime Darby Systems
Wisma Sime Darby
17th Floor, East Wing
Jalan Raja Laut
50350 Kuala Lumpur
Phone: 60 (3) 2939322
FAX: 60 (3) 2939376
Telex:
(784) 32807 SDCOMP MA

(Color Printers Only)
Uniphone Sdn. Bhd.
Apple Division
2nd Floor Annexe Block,
Menara Tun Rak
Jalan Raja Llaut
50350 Kuala Lumpur
Phone: 60 (3) 2918000
FAX: 60 (3) 2918280
Telex:
(784) 31152 PRECOM MA

MALTA

Fabian Enterprises Ltd.
18-20, Msida Road
Gzira
Phone: 356 (*) 31-32-83
FAX: 33 80 87
Telex: (838) 1837 FABIAN

MEXICO

Tektronix S.A. de C.V.
Viena 71, ler. Piso
Col. el Carmen, Coyoacan
04100 Mexico, D.F.
Phone: 52 (5) 658-7978,
659-4986
FAX: 52 (5) 658-7982

MOROCCO

(All products except CGG)
Tradec-Signal
Place Khaibar
Angle Bd de la Gironde et
Rue d'Ifni
Casablanca 03
Phone: 212 (*) 24 85 43
Telex: 25992

Worldwide Sales and Service Offices

THE NETHERLANDS

Tektronix Holland N.V.
Sales and Service Office
Planetenweg 99
2132 HL Hoofddorp
Mail:
P.O. Box 226
2130 AE Hoofddorp
Phone: 31(2503) 13300
FAX: 31(2503) 37271
Telex:
(844) 74898 TEKSO NL

NEW ZEALAND

Datamatic Pty. Ltd.
Wellington Office
8-10 Herbert St.
Lower Hutt Wellington
Mail:
P. O. Box 31-252
Lower Hutt,
New Zealand
Phone: 64 (4) 663-412
Toll Free: 0-800-65-5535
FAX: 64 (4) 664-630

Auckland Office
1 Woodson Place
Glenfield
Mail:
P.O. Box 34521
Birkenhead
Auckland
Phone: 64 (9) 443-7750
FAX: 64 (9) 443-6955

NIGERIA

Obodex Nigeria Ltd.
21A Airport Road
P.O. Box 55525
Ikeja, Lagos
Phone: (01) 962125, 962124
Telex:
(961) 26744 DATA NG

NORWAY

Tektronix Norge A/S
Trondheimsveien 436 B
Postboks 95 Grorudhagen
0905 OSLO 9
Phone: 47(2) 165050
FAX: 47(2) 165052

OMAN, SULTANATE OF

OHI Telecommunications LLC
P.O. Box 889
Muscat
Phone: 968 (*) 603-606, 603-607, 603-608
FAX: 968 (*) 603-608
Telex:
(498) 5052 TELECOM ON

PAKISTAN

Pakland Corporation (Pvt) Ltd.
Farhat Mansion
Iqbal Road
P.E.C.H. Society
Karachi 75400
Phone: 92 (21) 455-5384 455-5915
FAX: 92 (21) 43-7915
Telex:
(952) 25230 JALAL PK
Cable: 'PAKLAND' KARACHI

PANAMA

Executive Marketing Corp.
Ave El Paical, No. 30
Urbanizacion Los Angeles
Mail: Apartado 4929
Panama 5, Rep. de Panama
Phone: 507 (*) 36-0114 36-0722
FAX: 507 (*) 36-1820
Telex:
(328) 3483 ELECTRON PG

PERU

Importaciones y Representaciones Electronicas S.A. (IRE Ingenieros)
Avda. Franklin D. Roosevelt 105
Edificio Rimac
Lima 1
Phone: 51 (14) 288650
Fax: 51 (14) 271324
Telex: (394) 25663
IREING PU

PHILIPPINES

(Color Printers Only)
Distributed Processing Systems,Inc. (DPSI)
Don Chua Lamko Bldg.
H V dela Costa corner
Alfaro Sts
Salcedo Village, Makati,
Metro Manila
Phone: 63 (2) 818-9640, 818-9646, 818-9343
FAX: 63 (2) 817-4804
Telex: (742) 66284 DPSI PN

Philippine Electronic Ind., Inc.
3rd Floor,
Rose Industries Bldg
11 Pioneer Street
Pasig, Metro Manila
Mail:
P.O. Box 498
Makati, Metro Manila 1299
Phone: 63 (2) 631-1321 631-1323, 631-1324
FAX: 63 (2) 631-9333

POLAND

Tektronix Representation Office Warszawa
ul. Alzacka 15 A
PL-03-972 Warszawa
Phone: 48 (22) 17 28 50
FAX: 48 (22) 17 28 50

PORTUGAL

Decada Equipamentos de Electronica e Cientificos, SA
Rua Margarida Palla, 11 B
Miraflores/Alges
1495 Lisbon
Phone: 351 (1) 410-3420
FAX: 351 (1) 410-1844
Telex: (832) 15515 ESPEC P

QATAR

Business Communication Qatar
P.O. Box 3656
Doha
Phone: 974/32/58 51
FAX: 974/44 38 71
Telex: (497) 4219

Mideast Data Systems
P.O. Box 7899
Abu Dhabi
United Arab Emirates
Phone: 971 (2) 213 000
FAX: 971 (2) 213 741
Telex:
(893) 24 325 MDSUAE EM

ROMANIA

Tektronix Representative Office Bucuresti
Str. Uranus 98
Bloc U.8, Sc.2, Et.5, Ap 36
R-76102 Bucharest Sector 5

SAUDI ARABIA

Electronic Equipment Marketing Co. (EEMCO)
P.O. Box 3750
11481 Riyadh
Phone: 966 (1) 477 1650
FAX: 966 (1) 478 5140
Telex:
(928) 401120 ZUHAIR SJ

SINGAPORE

(Color Printers Only)
Diversitec Distributors PTE Ltd.
39 Sixth Avenue
Level 2, Guthrie Building
Singapore 1027
Phone: 65 (*) 468-3888
FAX: 65 (*) 469-8193
Telex:
(786) 26193 ROBOT RS

Mecomb Singapore Ltd.
Sime Darby Centre
896 Dunearn Road, 04-02
Singapore 2158
Phone: 65 (*) 469-8833
FAX: 65 (*) 467-1905, 468-0978
Telex: (786) 23178 RS

SLOVENIA

Commerce p.o.
Einspielerjeva 6
YU-61109 Ljubljana
Phone: 38 (61) 329 265
FAX: 38 (61) 329 171

SOUTH AFRICA

Protea Technology Ltd. Johannesburg
26 6th Street
Wynberg, Sandton 2090
Mail: Private Bag X19
Bramley 2018
Transvaal
Phone: 27 (11) 786-3647
FAX: 27 (11) 786-3789
Telex: (960) 4-24409 SA

Capetown
28 Main Rd
Rondebosch 7700
P.O. Box 23694
Claremont 7735
Phone: 27 (21) 685-4127/-5000
FAX: 27 (21) 689-4807
Telex: (960) 5-26555 SA

Durban
First Floor, Upminster
Essex Gardens, Nelson Road
Westville 3630
Mail: P.O. Box 1375
Wandsbeck 3631 Natal
Phone: 27 (31) 86-6324
FAX: 27 (31) 86-1932
Telex: (960) 6-20981 SA

SPAIN

Tektronix Española S.A. Madrid
Calle Condesa de Venadito, 1-5.planta
28027 Madrid
Phone: 34 (1) 404.1011
FAX: 34 (1) 404.0997
Telex: (831) 46014 TKME E

Barcelona
C/Diputación, 279, 6.planta
08007 Barcelona
Phone: 34 (93) 487 89 00
FAX: 34 (93) 487 41 33
Telex: (831) 97398 TKBE E

Bilbao Office
Villa de Plencia, 8 Bajo
48930 Getxo (Vizcaya)
Phone: 34 (4) 464.7822
FAX: 34 (4) 464.7066

SRI LANKA

Computerlink Data Systems, Limited
248 Galle Road
Colombo 4
SRI LANKA
Phone: 94 (1) 502111, 502142, 502143
FAX: 94 (1) 502203
Telex: (954)
22455 COLINK CE
21321 INLINK CE

SWEDEN

Tektronix AB Stockholm
Adolfsbergsvägen 31
Mail:
Box 20085
S-161 02 BROMMA
Phone: 46(8) 29 21 10
FAX: 46(8) 98 61 05

Gothenburg
Långedragsvägen 48
Mail: Box 4093
S-421 04 Västra Frölunda
Phone: 46 (31) 691080
FAX: 46 (31) 691522

SWITZERLAND

Tektronix International A.G.
Gubelstrasse 11
Mail:
P.O. Box 2142
CH-6300 Zug 2
Phone: 41(42) 219192
FAX: 41(42) 217784
Telex:
(845) 868808 TEKZ CH

TAIWAN

Tektronix Taiwan, Ltd Taipei
6th Floor, CDC Tower
125 Nanking East Rd, Sec 5
PO Box 36-659
Taipei 10572, Taiwan
Phone: 886 (2) 765-6362
FAX: 886 (2) 767-4360

Kaohsiung Office
2nd Floor-2
21 Chung-Hwa 3rd Rd.
Kaohsiung 80103
Phone: 886 (7) 215-2212
FAX: 886 (7) 215-2619

THAILAND

Dynamic Supply Engineering R.O.P.
Sukhumvit 63
Bangkok 10110
Phone: 66 (2) 392-5313 (8 lines)
FAX: 66 (2) 381-1467

Comgraph Co., Ltd.
199/1 Soi Sueksavittaya
North Sathorn Road
Bangkok 10500
Phone: 66 (2) 233-2679, 234-2782
FAX: 66 (2) 237-5011

TUNISIA

Societe El Eslek
3 Rue de la Monnaie
1001 Tunis
Phone: 216 (1) 244372, 342003
Telex:
(934) 13664 ESLEK TN

TURKEY

Inter Mühendislik Danismanlik Ve Ticaret A.S.
Hasircibasi
Caddesi No. 55
T-81310 Kadiköy-Istanbul
Phone: 90 (1) 349 94 00
FAX: 90 (1) 349 94 30
Telex:
(821) 29245 inmd tr

Inter Ankara
Hulya Sok Nr. 37/8
TR-067 00 GOP-Ankara
Phone: 90 (4) 43 64 421
FAX: 90 (4) 43 64 493
Telex: (821) 46780

UNITED ARAB EMIRATES

Mideast Data Systems U.A.E.
Ahmed Bin Ghannoum Bldg.
Salam Street
Mail:
P.O. Box 7899
Abu Dhabi
Phone: 971 (2) 213 000
FAX: 971 (2) 213 741
Telex:
(893) 24 325 MDSUAE EM

UNITED KINGDOM

Tektronix U.K. Limited
Fourth Avenue
Globe Park,
Marlow, Bucks SL7 1YD
Phone: 44 (0628) 486000
FAX: 44 (0628) 47 4799
Telex: (851) 847277, 847378 TEKMAR G

Scotland
Pentland House,
Almondvale South,
Livingston,
West Lothian, EH54 6NG
Scotland
Phone: 44 (506) 32766
FAX: 44 (506) 34974
Telex:
(851) 727056 TEKLIV G

UNITED STATES

Alabama

Huntsville:
4970 Corporate Drive,
Suite 100
Huntsville 35805
Phone: (205) 830-9212
FAX: (205) 837-2666

Alaska
Served by Frigid North Company (Representative), Anchorage, Alaska, (907) 561-4633

Arizona

Phoenix:
3015 S. 48th Street,
Suite 100
Tempe 85282
Phone: (602) 438-1011
FAX: (602) 437-9210
Mailing Address:
P.O. Box 29540
Phoenix 85038

Arkansas
Served by EnTest, Inc. (Representative), Dallas, Texas, (800) 955-0077

California

Walnut Creek:
1340 Treat Blvd., Suite 660
Walnut Creek 94596
Phone: (510) 932-4949
Fax: (510) 932-8596

Worldwide Sales and Service Offices

Irvine:
17052 Jamboree Blvd.
Irvine 92714
Phone: (714) 660-8080
TELEFAX: (GP1)
(714) 660-8080 X311
FAX: (714) 474-1490
Mailing Address:
P.O. Box 19523
Irvine 92713

Los Angeles:
21300 Erwin Street
Woodland Hills 91367
Phone: (818) 999-1711
FAX: (818) 340-9840
Mailing Address:
P.O. Box 8500
Woodland Hills 91365

Manhattan Beach:
3601 Aviation Blvd.,
Suite 1200
Manhattan Beach 90266
Phone: (213) 643-6605

San Diego:
8810 Rehco Rd.,
Bldg. D, Suite A
San Diego 92121
Phone: (619) 558-3221
FAX: (619) 558-3232

Santa Clara:
3003 Bunker Hill Lane
Santa Clara 95052-9937
Phone: (408) 496-0800
TELEFAX: (GP1)
(408) 496-0800 X137
FAX: (408) 988-6051

Colorado

Denver:
393 Inverness Drive South
Englewood 80112
Phone: (303) 799-1000
FAX: (303) 790-4754
Mailing Address:
P.O. Box 6979
Englewood 80155-6979

Connecticut
Served by the Boston, Massachusetts field office

Delaware
Served by the Philadelphia, Pennsylvania field office

District of Columbia
See MARYLAND

Florida

Orlando:
12051 Corporate Blvd.
Orlando 32817
Phone: (407) 249-1600
From the Cape Canaveral Area: (407) 636-0343
FAX: (407) 277-4723

Pensacola:
4700 Bayou Blvd., Bldg. 1
Pensacola 32503
Phone: (904) 476-1897
FAX: (904) 476-0720

Georgia

Atlanta:
Technology Park/Atlanta
650 Engineering Drive
Norcross 30092
Phone: (404) 449-4770
FAX: (404) 449-0358
Mailing Address:
P.O. Box 6500
Norcross 30091

Hawaii

Honolulu:
ComTest Technologies, Inc.
3049 Ualena St., Suite 1005
Honolulu, Hawaii 96819
Phone: (808) 831-0020
FAX: (808) 831-0017

Idaho
Served by Seltech, Inc. (Representative), Bellevue, Washington
(206) 746-7970

Illinois

Chicago:
5350 Keystone Court
Rolling Meadows 60008
Phone: (708) 259-7580
FAX: (312) 259-8388

Indiana

Indianapolis:
8751 Wesleyan Road
Indianapolis 46268
Phone: (317) 872-3708
FAX: (317) 875-0661

Iowa
Served by the Chicago, Illinois field office
Phone: (708) 259-7580 or Dallas, Texas field office
Phone: (800) 852-4043

Kansas

Kansas City:
Served by the Dallas, Texas field office
Phone: (800) 852-4043

Kentucky
Served by the Dayton, Ohio; St. Louis, Missouri; and Indianapolis, Indiana field offices

Louisiana
Served by EnTest, Inc. (Representative), Dallas, Texas, (800) 955-0077

Maine
Served by the Boston, Massachusetts field office

Maryland

DC:
700 Professional Drive
Gaithersburg 20879-3499
Phone: (301) 948-7151
FAX: (301) 921-0461
Mailing Address:
P.O. Box 6026
Gaithersburg 20884-0986

Massachusetts

Boston:
482 Bedford Street
Lexington 02173
Phone: (617) 861-6800
FAX: (617) 863-5994

Michigan

Detroit:
37890 Interchange Drive
Farmington Hills 48331
Phone: (313) 478-5200
FAX: (313) 478-1521

Minnesota

St. Paul:
2685 Long Lake Road
Building A
Roseville 55113
Phone: (612) 635-0520
FAX: (612) 635-0716

Mississippi
Served by EnTest, Inc. (Representative), Dallas, Texas, (800) 955-0077

Missouri

St. Louis:
1854 Lackland Hill Parkway
St. Louis 63146
Phone: (314) 569-1060
FAX: (314) 569-1061

Montana
Served by Seltech, Inc. (Representative), Bellevue, Washington
(206) 746-7970

Nebraska
Served by the Tektronix T&M National Marketing Center, Beaverton, Oregon
(800) 426-2200

Nevada
Served by the Walnut Creek, California and Albuquerque, New Mexico field offices

New Hampshire
Served by the Boston, Massachusetts field office

New Jersey

Woodbridge:
40 Gill Lane
Woodbridge 07095
Phone: (201) 636-8616
FAX: (201) 636-7234

New Mexico

Albuquerque:
6100 Uptown Blvd.
Suite 640
Albuquerque 87110
Phone: (505) 884-2828
FAX: (505) 884-8558

New York

Albany:
26 Computer Drive West
Albany 12205
Phone: (518) 458-7291
FAX: (518) 458-7361

Long Island:
225 Wireless Blvd.
Hauppauge 11788
Phone: (516) 231-9400
From NYC: (718) 895-8010
FAX: (516) 231-9624

Rochester:
1160 Pittsford-Victor Road,
Suite C
Pittsford 14534
Phone: (716) 383-0070
FAX: (716) 383-0022

North Carolina

Raleigh:
2950 Gateway Centre Blvd
Morrisville 27560
Phone: (919) 481-3800
FAX: (919) 469-8823

North Dakota
Served by the Tektronix T&M National Marketing Center, Beaverton, Oregon
(800) 426-2200

Ohio

Dayton:
8571 Gander Creek Drive
Miamisburg 45342
Phone: (513) 438-1166
FAX: (513) 438-1106

Oklahoma

Oklahoma City:
Served by the Dallas, Texas field office
Phone: (800) 852-4043

Oregon

Portland:
Mailing Address:
Portland Sales M/S 02-PTLD
P.O. Box 500
Beaverton 97077
Phone: (503) 627-7999
FAX: (503) 627-7995
Shipping Address
Portland Sales M/S:
02-PTLD
Regional Distribution Center
Beaverton 97077

Factory Service Center:
Tektronix Industrial Park
Beaverton 97077
Phone: (503) 627-6630
1-800-835-9433
TLX: 192825 TEKTRONIX

Pennsylvania

Philadelphia:
450 Sentry Parkway
Blue Bell 19422
Phone: (215) 825-6400
FAX: (215) 825-8839

Pittsburgh:
1051 Brinton Road,
Suite 300
Pittsburgh 15221
Phone: (412) 244-9800
FAX: (412) 244-1948

Rhode Island
Served by the Boston, Massachusetts field office

South Carolina
Served by the Atlanta, Georgia and Raleigh, North Carolina field offices

South Dakota
Served by the Tektronix T&M National Marketing Center, Beaverton, Oregon
(800) 426-2200

Tennessee
Served by the Atlanta, Georgia field office

Texas

Austin:
Served by the Houston and Dallas, Texas field offices

Dallas:
1551 Corporate Drive
Irving 75038
Phone: (214) 550-0525
Metro: (214) 751-0470
Sales FAX:
(214) 550-8226
Service FAX:
(214) 550-7363
Mailing Address:
P.O. Box 165027
Irving 75016

El Paso:
10432 Brian Mooney,
Suite 101
El Paso 79935
Sales:
(915) 593-8090
Service:
(915) 593-8095

Houston:
One West Chase Center
10777 Westheimer
Suite 675
Houston 77042
Phone: 1-800-847-0407
(713) 975-8670
FAX: (713) 784-4440

Kelly:
1831 S. General
McMullen Dr., Suite 101
San Antonio 78226
Phone: (512) 432-1341
FAX: (512) 432-1859

Utah
Served by the Seattle, Washington field office

Vermont
Served by the Albany, New York field office

Virginia

Crystal City:
Crystal Square 2
1725 Jefferson Davis Hwy.,
Suite 406
Arlington 22202
Phone: (703) 486-1300
FAX: (703) 486-1305

Hampton:
525 Butler Farm Road,
Suite 102
Hampton 23666
Phone: (804) 865-1588
FAX: (804) 865-7892

Washington

Seattle:
11814 115th Avenue NE
Kirkland 98034
Phone: (206) 821-9100
FAX: (206) 821-3607

West Virginia
Served by the Pittsburgh, Pennsylvania field office

Wisconsin
Served by the Chicago, Illinois, and St. Paul, Minnesota field offices

Wyoming
Served by the Tektronix T&M National Marketing Center, Beaverton, Oregon
(800) 426-2200

URUGUAY

Coasin Uruguaya S.A.
Libertad 2529
Mail:
Casilla de Correos 1400
Correo Central
Montevideo
Phone: 598 (2) 782000
FAX: 598 (2) 79-7338
Telex:
(398) 23010 COAUR UY

VENEZUELA

Equilab, C.A.
Caracas
Centro Sequros La Paz,
Piso 6, Ofic. E-61
Av. Francisco de Mirada
Boleita
Caracas 1070
Phone: 58 (2) 239-3233
FAX: 58 (2) 239-5205
Telex: (395) 21860 EQUIX

Maracaibo
Centro Comercial Montielco
Oficina 4, Piso 5
Avda 5 de Julio con calle 72
Maracaibo
Mail:
Apartado 1949
Maracaibo 4001-A
Phone: 58 (61) 52-4578,
52-7490
FAX: 58 (61) 51-2777
Telex: (395) 62154 EQUIX

YEMEN ARAB REPUBLIC

Al Qasr Trading Est.
P.O. Box 394
Sanaa
Yemen Arab Republic
Phone: 967 (2) 272 999
Telex: (895) 3117 NASRYE

YUGOSLAVIA

Commerce p.o.
Einspielerjeva 6
YU-61109 Ljubljana
Phone: 38 (61) 322-241
FAX: 38 (61) 329-540
Telex: (862) 31 283

Telcom-Panex
H.N. Zivkovica 2/IV
11000 Beograd
Phone: 38 (11) 185 834
FAX: 38 (11) 327 905

Tektronix
Representation Office
Belgrade
29, Novembra 15/V
YU-11000 Belgrade
Phone: 38 (11) 335 997
FAX: 38 (11) 323 941
Telex: (862) 11430

ZIMBABWE

Protea Medical Services (Pvt) Ltd.
111 Kelvin Road South
Graniteside
Mail:
P.O. Box 2766
Harare
Phone: 263 (4) 704094/9
FAX: 263 (4) 704652
Telex:
(987) 2-4102 PROTA ZW

Response card for International use, or call your local Tektronix Sales Office

I would like more information on: _____

_____

_____

Name _____

Title _____

Company/Mail Station _____

Address _____

City _____ Country _____ Postcode _____

Telephone _____ Ext. _____

Please send the following publications:

- ❑ **VXIbus Product Catalog** 11095, 11224
- ❑ **VXI Test Solutions Brochure** 11224, 11225
- ❑ **Designing and Evaluating VXI Systems** 11224, 11225, 11226
- ❑ **Open Measurement Systems Brochure** 11224, 11225, 11227

59W-283343

Response card for use within the United States, or call 1-800-426-2200 Ext. 132

I would like more information on: _____

_____

_____

Name _____

Title _____

Company/Mail Station _____

Address _____

City _____ State _____ Zip _____

Telephone _____ Ext. _____

Please send the following publications:

- ❑ **VXIbus Product Catalog** 11095, 11224
- ❑ **VXI Test Solutions Brochure** 11224, 11225
- ❑ **Designing and Evaluating VXI Systems** 11224, 11225, 11226
- ❑ **Open Measurement Systems Brochure** 11224, 11225, 11227

59W-283343

Place
Postage
Here

TEKTRONIX INC
P O BOX 1520
PITTSFIELD MA 01202-9864

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO 126 PITTSFIELD MA

Postage Will Be Paid By Addressee

TEKTRONIX INC
P O BOX 1520
PITTSFIELD MA 01202-9864

Tektronix Offers VXI Solutions – Not Just Components

Solving a test problem is not simply a matter of "catalog shopping." Certainly the technical requirements of the unit under test are as important as ever; they determine the basic functional nature of the test system. But the test system's impact on time to market, cost of test, and throughput are just as critical in selecting the right system.

To answer this larger need, your test system vendor must offer solutions, not just individual components. The definition of "solution" varies from situation to situation, but in effect it means providing everything needed to fully solve the problem: instruments, software, service offerings, training, applications help, configuration planning, and more.

With its broad product line, its Synergy program (a network of system integrators and distributors), and its alliances with such partners as Sun Microsystems, National Instruments, and Digital Equipment Corporation, Tektronix can deliver exactly the test solution you need. And of course, every Tektronix system is backed by the company's worldwide sales and support organization.

Call your local Tektronix representative for more information about system solutions. Sales office locations, U.S. and international, are listed at the end of this publication.

U.S.A., Africa, Asia, Australia, Central & South America, Japan
Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077-0001
For additional literature, or the address and phone number of the Tektronix Sales Office or Representative nearest you, contact: (800) 426-2200

Belgium: Brussels
Phone: 32(2) 725 96 10
FAX: 32(2) 725 99 53

Canada: Barrie
Phone: (705) 737-2700
FAX: (705) 737-5588

Denmark: Copenhagen
Phone: 45 44 53 54 55
FAX: 45 44 53 07 55

Eastern Europe, Middle East, and Austria
Tektronix Ges.m.b.H.
Doerenkampgasse 7
A-1100 Vienna, Austria
Phone: 43(222) 68-66-02-0
FAX: 43(222) 68-66-00

Finland: Helsinki
Phone: 358(0) 7282 400
FAX: 358(0) 7520033

France and North Africa
Tektronix S.A.
ZAC Courtaboeuf, 4 Av du Canada
B.P.13
91941 Les Ulis Cedex, France
Phone: 33(1) 69 86 81 81
FAX: 33(1) 69 07 09 37

Germany: Koeln
Phone: 49 (221) 96969-0
FAX: 49 (221) 96969-362

Italy: Milan
Phone: 39(2) 84441
FAX: 39(2) 8950-0665

Japan: Tokyo
Phone: 81(3) 3448-4611
FAX: 81(3) 3444-0318

The Netherlands: Hoofddorp
Phone: 31(2503) 13300
FAX: 31(2503) 37271

Norway: Oslo
Phone: 47(2) 165050
FAX: 47(2) 165052

Spain: Madrid
Phone: 34 (1) 404.1011
FAX: 34 (1) 404.0997

Sweden: Stockholm
Phone: 46(8) 29 21 10
FAX: 46(8) 98 61 05

Switzerland: Zug
Phone: 41(42) 219192
FAX: 41(42) 217784

U.K.: Marlow
Phone: 44 (0628) 486000
FAX: 44 (0628) 47 4799

Tektronix sales and service offices around the world: Algeria, Argentina, Australia, Austria, Bahrain, Bangladesh, Belgium, Bolivia, Brazil, Bulgaria, Canada, Chile, People's Republic of China, Colombia, Costa Rica, Cyprus, Czechoslovakia, Denmark, Ecuador, Egypt, Finland, France, Germany, Greece, Hong Kong, Iceland, India, Indonesia, Ireland, Israel, Italy, Ivory Coast, Japan, Jordan, Korea, Kuwait, Lebanon, Malaysia, Mexico, The Netherlands, New Zealand, Nigeria, Norway, Oman, Pakistan, Panama, Peru, Philippines, Poland, Portugal, Saudi Arabia, South Africa, Singapore, Spain, Sri Lanka, Sweden, Switzerland, Taiwan, Thailand, Tunisia, Turkey, United Arab Emirates, United Kingdom, Uruguay, Venezuela, Zimbabwe.

10/92 2 M 59W-7228-5

Tektronix